UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

# TESINA DE LICENCIATURA

Título: **Evolución Semántica en Wikis: una estrategia basada en refactorings**
Autor: **Etchevest, Mauricio Hernán**
Director: **Dra. Díaz, Alicia**
Carrera: **Licenciatura en Informática (plan 90)**

## Resumen

*Las **Wikis** han ganado popularidad gracias a la libertad y flexibilidad con la que permiten a los usuarios generar contenido. Por otro lado, las **Wikis Semánticas** extienden el concepto mediante la incorporación de anotaciones semánticas. Las anotaciones se realizan en forma de texto y se generan junto al contenido de los artículos wikis. El conjunto de todas las anotaciones conforman la ontología de la **Wiki Semántica**. La flexibilidad y libertad de las **Wikis Semánticas** provoca que el trabajo de los usuarios sea de forma descoordinada. Esta forma de trabajo provoca que la ontología desarrollada sea de baja calidad afectando la navegabilidad, la exactitud de las búsquedas y la precisión de la ontología. En esta tesis se desarrolla una estrategia para mejorar la calidad de la ontología. La estrategia está basada en dos conceptos claves: los **Semantic Wiki Bad Smells** que representan un error en la estructura interna de una **Wiki Semántica** y los **Semantic Wiki Refactorings** que representan procesos necesarios para eliminar total o parcialmente un **Bad Smell**. La presencia de un **Semantic Wiki Bad Smell** no indica necesariamente un error, sino que es un síntoma que debe ser analizado en su contexto. Ese análisis debe ser realizado por los usuarios ya que conocen el dominio de la wiki. Se asiste al usuario en las diferentes etapas de evolución para mejorar la calidad de la ontología de una **Wiki Semántica**.*

## Palabras Claves

*Wiki, Wiki Semántica, Bad Smell, Refactoring, Media Wiki, Semantic Media Wiki, Ontología, Evolución en Wikis Semánticas, Semantic Wiki Bad Smell, Semantic Wiki Refactoring.*

## Conclusiones

*Se desarrolla una estrategia para asistir al usuario en la continua evolución semántica en **Wikis Semánticas**. La estrategia incluye las herramientas para analizar el estado de la ontología, y posiblemente realizar modificaciones. También se brinda al usuario información necesaria para tomar las decisiones adecuadas y evaluar cuando aplicar un **refactoring.** De esta forma se espera mejorar la experiencia de los usuarios con Tecnologías Semánticas.*

## Trabajos Realizados

*El principal aporte realizado es mejorar la experiencia de los usuarios en la utilización de tecnologías semánticas. Proveer una herramienta para asistir al usuario durante todas las etapas de la evolución. Asociar los dos conceptos claves de la estrategia, los **Semantic Wiki Bad Smell** y **Semantic Wiki Refactorings**. Para cada uno de los **bad smells** se indica los **refactorings** que elimina el síntoma. El desarrollo de un catálogo de **Semantic Wiki Bad Smells** y **Semantic Wiki Refactorings**. La herramienta **Semantic Grouper** para la creación de conjuntos de forma dinámica.*

## Trabajos Futuros

*El principal requerimiento es la investigación y desarrollo de nuevos **Semantic Wiki Bad Smells** y **Semantic Wiki Refactorings**. Desarrollar nuevas herramientas como la capacidad de realizar **UNDO** (o deshacer), que consiste en llevar a la ontología al estado anterior a la ejecución de un **refactoring**. El **Garbage Collector**, para eliminar de forma definitiva y automática los artículos wikis que la herramienta elimina de forma lógica. Extender el comportamiento del **Semantic Grouper** para desagrupar conjuntos de forma automática. Realizar un historial de los **refactorings** aplicados a una ontología.*

**Fecha de la presentación:** Febrero 2013

# Índice

# Capítulo 1

## Introducción

*Ged, escuchame ahora..¿Nunca has pensado que así como hay oscuridad alrededor de la luz, también hay peligro alrededor del poder?.*

***Ursula K. Le Guin.***

## Introducción.

Las ***Wikis*** son aplicaciones web que facilitan la creación de contenido de forma colaborativos. Se caracterizan por la libertad y flexibilidad con la que permiten a los usuarios generar contenido. Surgen para satisfacer la necesidad de generar contenido de forma colaborativa pero rápida y dinámica.

Las ***Wikis*** tradicionales son aplicaciones ampliamente exitosas. Actualmente se comienzan a utilizar las ***Wikis Semánticas*** que extienden a las ***Wikis*** tradicionales. Una ***Wiki Semántica*** combina las propiedades de las **Wikis** tradicionales con las tecnologías de la **Web Semántica**

Los usuarios de las ***Wikis Semánticas*** agregan anotaciones semántica a los *artículos wikis*. Luego, el conjunto de todas las anotaciones semánticas definen su *ontología* subyacente. En el proceso de creación de la *ontología* surgen diferentes problemas provocados por la forma colaborativa de su desarrollo.

En este capítulo, se describen los problemas que se encuentran en el desarrollo de una *ontología* y se propone una estrategia para asistir al usuario durante la evolución de la ***Wiki Semántica***. La estrategia esta basada en dos conceptos claves, los ***bad smells*** y en ***refactorings.***

## 1. Motivación.

Una ***Wiki*** [1] es una aplicación web que permite a múltiples usuarios la creación y edición de páginas web de forma colaborativa. La característica más importante de una ***Wiki*** es la libertad y flexibilidad con la que permite generar contenido a los usuarios. Las ***Wikis*** proveen un editor de texto web a través del cual los usuarios utilizan un sistema de *tags* para la creación de *links*, darle estilo al texto o categorizar los artículos.

Las ***Wikis Semánticas*** [2] extienden el concepto de una ***Wiki*** tradicional relacionando el contenido textual con información estructurada. Para lograr esta relación, soportan meta información en forma de *anotaciones semánticas* (o *tag semánticos*) dentro de los artículos de la ***Wiki***. Las ***Wikis Semánticas*** han ganado considerable atención como conectores entre la '*inteligencia social'* y la *'inteligencia artificial*' proporcionando una simple introducción a la nueva '**Web Semántica**' siendo las ***Wikis Semánticas*** una de las aplicaciones mas exitosas.

Entre las ventajas de utilizar una ***Wiki Semántica*** se obtiene la capacidad de deducir información derivada, realizar búsquedas por relaciones entre objetos, obtener un formalismo en la anotación de artículos y otro tipo de contenido generando así más información para las búsquedas, o el intercambio de información entre diferentes aplicaciones.

Los usuarios crean y mantienen la *ontología* de una ***Wiki Semántica*** asociando un concepto de un *artículo wiki* con la *ontología*. Utilizando *links* y anotaciones se relacionan los conceptos y los artículos entre sí. Una anotación define que valor tendrá el *artículo wiki* para una determinada propiedad.

Dentro de una ***Wiki Semántica*** el conjunto de todas sus anotaciones semánticas definen su *ontología* subyacente. Informalmente, una *ontología* es una especificación explícita y formal de una conceptualización [2]. Mas precisamente, una *ontología* es un modelo formal de un dominio que describe sus conceptos y las relaciones entre ellos.

Las ***Wikis*** evolucionan como resultado del trabajo colaborativo de sus usuarios, consecuentemente cuando la ***Wiki*** es semántica, la *ontología* emerge como resultado del esfuerzo colaborativo de sus usuarios. Dentro de una ***Wiki Semántica*** los usuarios tienen la libertad y flexibilidad para definir anotaciones semánticas.

Sin embargo, esta forma de trabajo genera que los usuarios trabajen de forma descoordinada y puedan generar una *ontología* de baja calidad. Al no contar con una *ontología* de buena calidad, se afecta la navegabilidad de la ***Wiki***, la exactitud de las búsquedas semánticas y el nivel de precisión de la *ontología, así* decrementando la usabilidad de la ***Wiki***.

Esto marca la necesidad de contar con herramientas que faciliten la evolución de la *ontología* con el objetivo de mejorar su calidad. Siendo los usuarios finales los encargados de utilizar la herramienta ya que son los expertos en el dominio.

A fin de comprender las características del problema se presenta un ejemplo para el cual se requiere solución.

*Supongamos que en las etapas iniciales de una **Wiki** se crea una categoría llamada 'Ciudad'. Luego, otro usuario crea la categoría 'Ciudad Capital' intentando formar un conjunto mas específico de ciudades que también son capitales. Sin embargo, el usuario que crea la categoría 'Ciudad Capital' no realiza la especificación que indica que la categoría es un subconjunto de 'Ciudad'. La falta de la información semántica sobre esa relación de los dos conjuntos trae diferentes problemas. Para comenzar afecta la consistencia de la **Wiki Semántica**. Cada vez que un usuario crea un artículo de una ciudad que también es ciudad capital, deberá categorizar el artículo con las dos categorías para no perder información semántica. También afecta la completitud, un usuario consultando la **Wiki** por una lista de ciudades, no va a obtener como resultado aquellos artículos que solo contengan la categorización de 'Ciudad Capital'.*

En esta tesis proponemos utilizar una estrategia basada en ***refactorings*** para asistir al usuario en la evolución semántica de las ***Wikis Semánticas***.

**2. Objetivo.**

Desarrollar una estrategia para asistir al usuario en la mejora continua de la calidad de la *ontología* subyacente en una ***Wiki Semántica***. Mejorando la calidad de la *ontología* se espera mejorar la navegabilidad de la ***Wiki***, optimizar los resultados de las consultas semánticas y mejorar la consistencia.

Por las características del problema, se adaptará la técnica de ***refactoring*** para producir cambios en la *ontología* subyacente de una ***Wiki Semántica*** y así mejorar su calidad. Con este objetivo se utiliza la técnica de ***refactoring*** de la *Ingeniería de Software* [3] para su utilización en las ***Wikis Semánticas***.

La técnica de ***refactoring*** consiste en detectar posibles errores mediante ***bad smells***. Luego, para producir los cambios y mejora de la *ontología* se utilizarán los ***refactorings***. De cada ***refactoring*** se contará con un listado ordenado de modificaciones que debe ser producidas en la ***Semantic Wiki*** para eliminar total o parcialmente el ***bad smell***.

Al desarrollar una estrategia basada en ***refactorings***, se necesita tener una definición formal de los ***refactorings*** y ***bad smells*** conocidos. De esta forma el usuario podrá ver cuales son los efectos de la ejecución de los ***refactorings*** y ***bad smells***.

**3. Desarrollos Propuestos.**

Se propone desarrollar una herramienta en forma de extensión de ***Media Wiki*** [6] para ser utilizada junto con la extensión **Semantic Media Wiki (SMW)** [5], dado que es la ***Wiki Semántica*** más utilizada.

Se adaptará la técnica de ***bad smells***, que permite la detección de los posibles errores. Para facilitar su utilización, siempre que se pueda, se definirá una consulta semántica para realizar la detección automática, de no ser posible se realizará de forma semi-automática.

Como parte de la estrategia de ***refactorings*** se realizará un catálogo de ***Semantic Wiki Refactorings*** y

de ***Semantic Wiki Bad Smells***. Este catálogo tendrá la definición formal de estos, y la relación de cada ***bad smell*** con uno o varios ***refactoring***.

**4. Organización de la tesis.**

En el *capítulo 2* se describe el estado actual de las tecnologías involucradas en esta tesis. En el *capítulo 3* se presenta la estrategia que se propone para cumplir con los objetivos planteados. En el *capítulo 4* se describe la implementación de la estrategia. Finalmente, en el *capítulo 5* se describen las conclusiones, con los aportes realizados y el trabajo a futuro.

# Capítulo 2

## Estado del arte

*The question of whether Machines Can Think... is about as relevant as the question of whether Submarines Can Swim.*

***Edsger W. Dijkstra.***

**Introducción.**

Una ***Wiki*** es una aplicación web que permite a múltiples usuarios la creación y edición de páginas web de forma colaborativa. Hasta la aparición de las ***Wikis*** las páginas web se creaban unicamente en **HTML**. Utilizando una ***Wiki*** los usuarios crean el contenido de una forma fácil y flexible con la capacidad de publicar el contenido de forma inmediata. Permiten crear páginas web de forma masiva y colaborativa donde todos los usuarios aportan conocimiento.

Los usuarios muchas veces perciben que las aplicaciones web son poco inteligentes. Esta percepción proviene de disponer información inconsistente, desincronizadas o desconectada. Las aplicaciones web son inteligentes pero no cuentan con la información necesaria para desarrollar su potencial. Una aplicación web es tan inteligente como la información disponible se lo permite. La ***Web Semántica*** tiene como objetivo proveer a las aplicaciones web información semántica en forma de meta-datos. Brindando una infraestructura de datos a las aplicaciones web se podrá mejorar la experiencia del usuario.

La ***Web Semántica*** tiene como objetivo mejorar la experiencia de los usuarios en las aplicaciones web. Se describe el lenguaje ***RDF (Resource Description Framework)*** que tiene como objetivo la representación la información requerida por la ***Web Semántica***. En ciertas aplicaciones la expresividad de ***RDF*** no alcanza. Por ese motivo se presenta el lenguaje ***OWL (Web Ontology Languaje)*** que intenta obtener un lenguaje con un balance entre la expresividad y la eficiencia de razonamiento.

Las ***wikis semánticas*** combinan las propiedades de las ***Wikis*** tradicionales con las tecnologías de la ***Web Semántica***. Para realizar una asociación entre una estructura y el contenido de un *artículo wiki*, los usuarios agregan anotaciones semánticas al contenido. Al contar con una estructura los usuarios obtienen beneficios como búsquedas basadas no solo en palabras claves sino en las diferentes relaciones que tienen las páginas. Se describe como funcionan y cuales son los beneficios que obtienen los usuarios al usar una ***wiki semántica***.

***Semantic Media Wiki (SMW)*** es una implementación de una ***Semantic Wiki*** siendo una de las más utilizadas en la actualidad, es un motor ***wiki*** mejorado semánticamente. Su principal objetivo es soportar *'Semantic Wikipedia'* por lo cual se tiene como objetivo la usabilidad y escalabilidad.

En este capítulo se describen las tecnologías involucradas en esta tesis. Se define el concepto de ***Wiki***. Luego se describe la ***Web Semántica*** junto con las tecnologías y lenguajes que utiliza para poder definir el concepto de ***Wiki Semántica***. Seguidamente se describe una implementación de una ***Wiki Semántica*** llamada ***Semantic Media Wiki***.

## 1. Wiki.

Una ***Wiki*** es una aplicación web que permite a múltiples usuarios la creación y edición de páginas web de forma colaborativa. La característica más importante de una ***Wiki*** es la libertad y flexibilidad con que permite generar nuevo contenido. Las ***Wikis*** proveen un editor de texto web donde los usuarios utilizan un sistema de *tags* para la creación de *artículos wikis* utilizando *links*, estilos de texto o categorizar las páginas.

La primer ***Wiki*** fue llamada WikiWikiWeb siendo desarrollada en el año 1995 por Ward Cunningham. Quien es considerado un precursor en el desarrollo de nuevos métodos, como programación orientada a objetos, diseño de patrones o *extreme programming*. Su motivación al desarrollar las ***Wikis*** fue mejorar los procesadores de textos convencionales para utilizar la nueva tecnología como mecanismo de documentación en el desarrollo de software. Su objetivo fue desarrollar un software para programadores pero relativamente sencillo que permita el trabajo colaborativo y publicar el contenido de forma inmediata. La palabra '*Wikiwiki*' se toma del idioma *Hawaiano* y significa 'rápido' o 'apurado'. El nombre describe la forma característica de las ***Wikis*** donde el contenido puede ponerse disponible de forma rápida y sin complicaciones.

En un principio las ***Wikis*** fueron desarrolladas para programadores, pero debido a su éxito actualmente son utilizadas por un amplio público en diferentes áreas. La forma de aplicación depende del objetivo de la comunidad u organización que utiliza la ***Wiki***. Se distinguen principalmente dos grupos de aplicaciones donde la diferencia es a quien está dirigido la ***Wiki***. El primer tipo de aplicaciones son las orientados a grupos cerrados de usuarios que lo utilizan para coordinar trabajos, o como *content management system (CMS)*. El segundo tipo de aplicación es cuando los usuarios son potencialmente todas las personas que acceden a la ***Web***.

Existe una amplia variedad de ***Wikis*** exitosas orientadas a toda la comunidad de la ***Web***. Uno de los mas conocidos es *Wikipedia* [ http://http://www.wikipedia.org/ ] creada en el año 2001 y consiste en una enciclopedia donde todos los usuarios aportan conocimiento. Otro caso es *WikiHow* [http://http://www.wikihow.com] donde la comunidad explica como realizar las más amplia variedad de tareas. Existen ***Wikis*** con diferentes tópicos, otro caso exitoso es *Wikitravel* [http://http://wikitravel.org] una guía de viajes internacional o *Wiktionary.org* [http://wiktionary.org/] un diccionario *online*.

Las ***Wikis*** permiten que los usuarios generen nuevo contenido desde el editor web y publicarlo en la web inmediatamente. El contenido está disponible para que otros usuarios realicen modificaciones o agregar contenido. Cualquier usuario puede crear o editar una *página wiki*. Esta libertar y flexibilidad permite que si un usuario crea contenido erróneo, otro usuario puede editar el contenido corrigiendo el error. Cuando los usuarios crean una nueva página el título de la página es el identificador, por lo cual no puede existir dentro de una misma ***Wiki*** dos artículos con el mismo título. Las páginas web creadas con una ***Wiki*** son comúnmente llamadas *páginas wikis o artículos wikis*.

Los usuarios son los encargados de crear las *páginas wiki* desde el editor de la ***Wiki***. En el proceso de creación se puede agregar al texto ciertos estilos o formatos al texto. Para esto se cuenta con un sencillo lenguaje de *mark up*. Se utilizan *tags* para definir el estilo de las fuentes o dar formato al texto.

En la siguiente imagen vemos un artículo que describe la ciudad de 'La Plata':


Imagen 1. Artículo de 'La Plata' en Wikipedia.org

En la imagen 1 vemos un artículo que describe la ciudad de 'La Plata'. Si un usuario piensa que puede realizar un aporte, debe ir a la solapa editar para realizar una edición al artículo.

En la siguiente imagen vemos como un artículo de Wikipedia puede ser editado simplemente accediendo a la solapa 'editar' del *artículo wiki*.

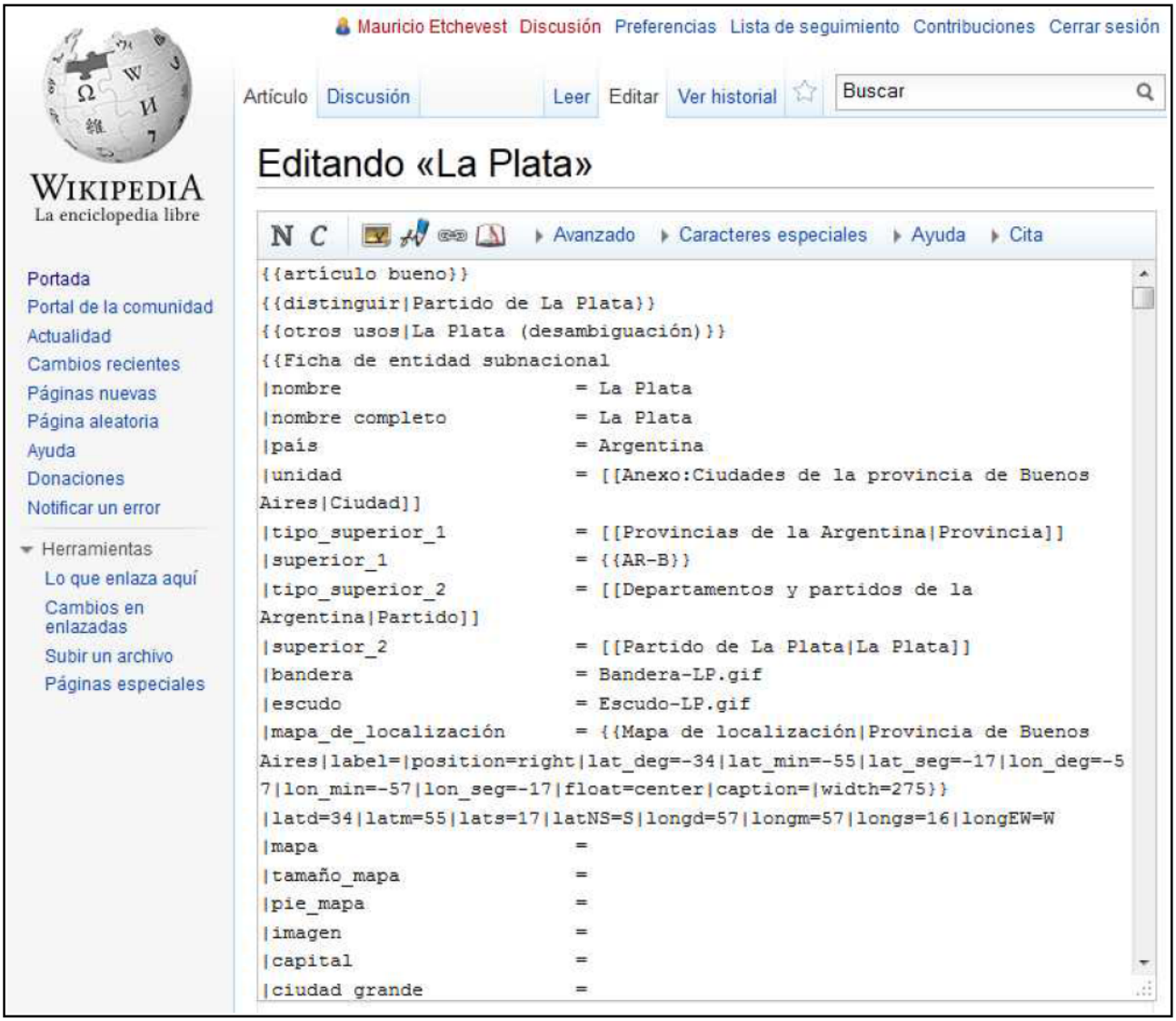


Imagen 2. Edición de un artículo en Wikipedia.

En la imagen 2 vemos el lenguaje de *mark up* que utilizan los usuarios para describir un *artículo wiki*. Una vez que realizan las ediciones deseadas sencillamente puede guardar el artículo y ver las modificaciones. En el historial del *artículo wiki* se almacenará una versión antes de la edición realizada.

Los usuarios crean los *artículos wiki* utilizando el editor web y describen el *artículo wiki* utilizando un lenguaje de *mark up*. Cuando se concluye con la creación del artículo es guardado en la ***wiki***. A partir de la descripción del *artículo wiki* en lenguaje *mark up* (imagen 2) el sistema ***wiki*** genera la versión de lectura del *artículo wiki* (imagen 1). A partir de un sencillo lenguaje de *mark up* el sistema ***wiki*** genera una página web.

Se puede relacionar *páginas wiki* utilizando un *link*. Un *link* es un acceso directo hacia otra *página wiki* que puede existir o no. En ciertos sistemas ***wikis*** un *link* se realiza poniendo entre corchetes '[' el nombre del artículo que deseamos enlazar. Cuando este artículo es visualizado se procesa el *link* y se muestra como un *link*

en **html** donde los usuarios pueden acceder. Cuando realizamos un enlace hacia otro *artículo wiki* puede suceder que el artículo exista. Si el artículo existe el usuario lo visualiza y tiene la posibilidad de editar su contenido. Si el artículo no existe, puede significar que se requiere la generación de nuevo contenido. Nuevamente el usuario puede generar un nuevo contenido utilizando el editor.

En la siguiente imagen vemos como al no encontrar una *artículo wiki*, lo podemos crear simplemente accediendo a la solapa 'Crear'. El mismo sistema ***Wiki*** nos aconseja estar seguros de que el artículo no existe, y en efecto es necesario la creación de un nuevo artículo.


Imagen 3. Artículo no encontrado.

Una ***Wiki*** es una aplicación colaborativa donde el contenido cambia de forma dinámica. Para mantener registro de los cambios realizados se cuenta con un sistema de versionado. Cada vez que se realiza una modificación en el artículo se resguarda la versión anterior. De esta forma al mantener todas las modificaciones que se realizaron, si un usuario intencionalmente decide agregar un dato erróneo, no perdemos el dato original y fácilmente se puede solucionar el error. Utilizando el versionado los usuarios pueden rastrear como fueron las sucesivas modificaciones realizadas al contenido de un determinado *artículo wiki*.

En la siguiente imagen vemos cuales fueron las ultimas modificaciones realizadas al *artículo wiki* que describe la ciudad de 'La Plata'.


Imagen 4. Cambios realizados al artículo 'La Plata'.

El cambio dinámico de las ***Wikis*** y la creación de contenido de forma constante provoca que las búsquedas sean un aspecto primordial en el éxito de las ***Wikis***. Para acceder a los contenidos los usuarios realizan búsquedas basadas en palabras clave. Las consultas basadas en palabras claves muchas veces no alcanzan para encontrar la información que se requiere. Existen ciertas relaciones entre la información contenida en una ***Wiki*** que no puede ser utilizada en estas consultas. Muchas veces los usuarios desearían realizar consultas basados en información estructurada existente en la ***Wiki***. Por ejemplo, realizar búsquedas basados en el lugar y fecha de nacimiento de una persona, o ubicación geográfica de una ciudad.

Para intentar solucionar este problema las ***Wikis*** generalmente tienen dos estrategias. La primera consiste en mantener manualmente información estructurada. Se podría contar con una página con todas las personas nacidas en una determinada fecha o incluso todos los acontecimientos importantes que sucedieron en una determinada fecha. Esta forma de organizar la información agrega mucho trabajo de mantenimiento, generalmente realizado por usuarios administradores o con privilegios diferidos.

Un ejemplo de las estructuras utilizadas para organizar la información son los *infobox*. Un *infobox* es una estructura fija en forma de tabla diseñada para lograr una vista mas estructurada de los *artículos wiki*. Es una tabla donde se presenta un resumen de los *artículos wiki* con la información más importante. Dado un conjunto de artículos que describen el mismo tipo de concepto se define un *infobox* con una estructura definida. Cuando se crea un nuevo *artículo wiki* se debe definir la información que se describe en el *infobox* definido para el tipo de artículo. Esta forma de presentar la información facilita incluso la navegación por las diferentes secciones de los artículos.

En la siguiente imagen se presentan un *infobox* utilizado en Wikipedia.org para describir personas:


Imagen 5. *Infobox* del artículo personal de **Edsger Wybe Dijkstra**.

Como se puede ver al definir un artículo de una persona se debe definir el nombre, información sobre

su nacimiento, su fallecimiento (si es que ocurrió), junto con otros datos importantes del artículo. Los campos son variables y depende de cuanta información contenga el artículo.

A continuación vemos otro *infobox* que contiene más información. Esto es debido a que el artículo donde se utiliza el *infobox* contiene más información que el anterior:


Imagen 6. *Infobox* del artículo personal de **Alan Turing**.

El segundo acercamiento es proveer herramientas para ordenar la información dentro de la ***Wiki***. Una de estas herramientas es la organización de los *artículos wiki* de acuerdo a categorías. Una categoría agrupa artículos con contenidos referentes a tópicos relacionados. Los usuarios definen a cuál o cuales categorías pertenece el artículo. Los usuarios agregan un artículo a una categoría utilizando el lenguaje de *mark up* mencionado anteriormente. Para agregar un *artículo wiki* a una categoría debemos agregar al artículo '[[ categoria : nombreCategoria ]]'.

En la siguiente imagen vemos las categorías a las que pertenece el artículo de *Edsger Wybe Dijkstra*. Al pie de cada artículo se pueden ver cuales son las categorías a las que pertenece.


Imagen 7. Categorías a las que pertenece al artículo de **Edsger Wybe Dijkstra**

Cada categoría cuenta con un artículo que describe la categoría (que artículos deben formar parte de la misma) y se muestran todos los artículos que pertenecen a la misma. La descripción debe ser utilizada por los usuarios para determinar que artículos deben pertenecer a la categoría. Al contar con una página que agrupa todos los artículos de una misma categoría se puede navegar los diferentes artículos de una categoría. Es decir, la categorización de artículos también facilita la navegabilidad de la ***wiki***.

Las ***wikis*** proveen otra herramienta para la organización de la información llamada *template*. Un *template* es un artículo creado para ser incluido dentro de otros artículos. Por lo general se crea un *template* con información que se repite en varios artículos. Los usuarios agregan el contenido de un *template* a un artículo agregando el texto '{{ nombre del *template* }}'. Un *template* también puede tener parámetros que luego se reemplaza el valor pasado por parámetro dentro del contenido del *template.*

Por ejemplo, se puede definir un *template* para definir un formato de fecha llamado 'FechaConEdad'.

Dentro del artículo donde queremos utilizar el *template* agregamos el siguiente texto:

{{ FechaConEdad | 06 | 05 | 1993 }}

Luego, este texto es reemplazado por el contenido del *template* y dentro del artículo se vería de la siguiente manera:

'6 de mayo de 1993 (19 años)'

Las soluciones que plantean las ***Wikis*** tradicionales no siempre cumplen con las expectativas. Los beneficios obtenidos no siempre superan al tiempo que los usuarios deben tomarse para aprender a utilizar estas herramientas. Para el éxito de una herramienta es esencial que los usuarios obtengan beneficios que justifiquen el tiempo de trabajo extra.

## 2. Semantic Web.

El concepto de ***Web*** consiste en la idea de una comunidad abierta donde cualquier persona puede contribuir con sus ideas, esta libertad logra generar una gran variedad de tópicos. Una ***Web*** de información es una entidad orgánica que crece con los intereses y energía de la comunidad que la soporta. Suele suceder que los usuarios perciben que la ***Web*** contiene mucha información pero no debidamente relacionada, o que no se encuentra información específica. Por esta razón se dice que la ***Web*** es inmensamente grande pero poco profunda.

Supongamos que queremos asistir a una conferencia en una ciudad a la cuál nunca visitamos. Accediendo a la página web de la conferencia y encontramos la dirección donde será llevada a cabo la conferencia. Luego, nos interesa reservar un hotel para nuestra estadía. Accedemos a la página web de una cadena de hoteles que está presente en la ciudad. De forma sencilla accedemos a un listado de las sucursales de la cadena de hoteles en la ciudad. Viendo cada una de las sucursales con su correspondiente dirección nos preguntamos ¿cuál es la sucursal de la cadena de hoteles que mas cerca está a la conferencia? ¿qué tan lejos queda el hotel del lugar de la conferencia?. Para responder estas preguntas debemos acceder a otra página web que cuenta con un mapa y una forma de calcular las distancias entre los dos puntos. Para esto debemos proporcionarle las direcciones del origen y el destino. Para esta tarea debemos copiar y pegar todas las direcciones de todas las sucursales de la cadena de hoteles junto con la dirección de la conferencia y anotar las distancias para comparar los diferentes hoteles. Mientras copiamos y pegamos todas las direcciones nos preguntamos ¿porque debemos ser nosotros los que hagamos todo este trabajo? Toda la información está contenida en la web, y también utilizamos una aplicación web para realizar los cálculos. ¿No debería haber una forma automática de realizar todos estos cálculos?.

Ahora supongamos que estamos investigando sobre el sistema solar. Realizando la investigación en la web encontramos una página donde se explica claramente los objetos del Sistema Solar. Se describen los planetas, las lunas, los asteroides, etc. Cada uno de los objetos tiene su propia página web donde se detalla la información sobre cada objeto e incluso se muestran fotos. Dentro de los detalles de cada objeto se ve una categorización del mismo. Para cada uno se describe si corresponde a un planeta, luna, asteroide o cometa. Otra página del mismo sitio web propone ver la información en forma de listados para su mejor comprensión. Con lo cuál contamos con listados de todos los planetas alrededor del sol, las lunas de Júpiter, los objetos con nombre del cinturón de asteroides, etc. La lista de los planetas alrededor del sol cuenta con nueve nombres familiares, cada uno enlazado con la página web propia con información detallada. Un día vemos en el diario de la *International Astronomical Union (IAU)* que se ha decidido que Plutón, que desde el año 2006 ha sido considerado un planeta, ahora debe ser considerado miembro de una nueva categoría llamada 'planeta enano'. Por curiosidad nos dirigimos a la página web de Plutón para ver si la información ha sido actualizada. Para nuestra sorpresa, la página web ya está actualizada, vemos en la página web de Plutón aparece en la categoría 'planeta enano'. Luego, nos dirigimos a la página donde se listan todos los planetas y vemos que Plutón todavía esta listado junto con el resto de los planetas. La página web fue actualizada de forma inconsistente, se modificó en los detalles de la página de Plutón pero no en la página que lista los planetas del sistema solar.

En los ejemplos anteriores vemos que las aplicaciones tienen representación de la información que los usuarios las perciben como poco inteligentes. La información de las páginas web se encuentran inconsistente, desincronizada y desconectada.

La situación es que las aplicaciones web son inteligentes pero no disponen de la información adecuada. Una aplicación web es tan inteligente como la información con la que cuenta. Si las aplicaciones web cuentan con la información para tomar las decisiones o la información está debidamente conectada, desde el punto de viste del usuario la aplicación es mas inteligente. Para lograr que las aplicaciones web sean inteligentes y conectadas es necesario contar con una infraestructura web que les permita obtener la información de forma adecuada.

En el ejemplo de la conferencia necesitamos que la información de las direcciones sean comprendidas de forma automática por la aplicación de mapas. En este aspecto se dice que las aplicaciones deben tener la información necesaria, la aplicación de mapas debe tener las direcciones de los lugares de origen y destino. La representación de las direcciones se encuentra en formato de texto, solo comprensible por los humanos. No debería ser necesaria la interpretación de la información por un ser humano para mover la información de un sitio al otro. La aplicación de mapas tiene la capacidad de calcular la ruta mas corta y la distancia necesaria, pero solo lo puede calcular si cuenta con la información adecuada.

En el ejemplo del Sistema Solar se requiere la actualización de forma consistente. Si la categoría de Plutón es 'planeta enano' se tendría que eliminar a Plutón de la lista de los planetas del Sistema Solar. No existe la representación del estado de un objeto como un planeta. En este aspecto la información debe estar conectada, la información debe ser consistente.

El objetivo es representar la información de forma que describa el objeto real y sus relaciones. La información de los ejemplos anteriores está representada de forma conveniente para su presentación. Es decir, se tiene la información de forma adecuada para la interpretación por un humano, pero no para una computadora.

En la actualidad se intenta no contar con la presentación de la información como la representación primaria de los datos. No se cuenta con una colección de páginas web sino que con una colección de datos, a partir de los cuales se genera la presentación. La aplicación no se centra en la presentación sino en los sujetos de la presentación. En este sentido es que las aplicaciones son semánticas, se representa de forma explícita las relaciones que subyacen a la aplicación y se generan las presentaciones cuando es necesario. Para este objetivo se construye una *ontología.* Una *ontología* es la formulación de un exhausto y riguroso esquema conceptual dentro del dominio de datos. La *ontología* tiene el objetivo de facilitar la comunicación y el intercambio de información entre diferentes sistemas y entidades. De esta forma se representa el conocimiento que tienen las aplicaciones del dominio al cuál pertenecen.

La infraestructura actual de la ***Web*** soporta una red distribuida de páginas web que se pueden referenciar unas a otras utilizando *links* llamados *Uniform Resource Locators (URLs)*. Ciertos sitios avanzados reemplazan esta estructura local con un respaldo de una base de datos o **XML** que asegura consistencia en las páginas. La principal idea de la ***Web Semántica*** es soportar una ***Web*** distribuida al nivel de datos en lugar de un

nivel de representación. En lugar de contar con una página web apuntando hacia otra, se tiene un ítem de datos apuntando a otro ítem. Para lograr esto se utilizan las referencias globales de *Uniform Resource Identifiers (URIs)*.

La infraestructura de la ***Web*** provee un modelo de datos que permite a la información acerca de una entidad simple esté distribuida a lo largo de la ***Web***. La información está presente en más de una página web controladas por diferentes organizaciones. Sin embargo el modelo de datos para la aplicación no está dentro de una de las aplicaciones sino que es parte de la infraestructura de la ***Web***.

En el ejemplo de la conferencia se publica en la página web información estructurada sobre la dirección donde será llevada a cabo. Junto con la representación para ser visualizada se publica una descripción de la información que pueda ser procesada por una computadora. Tanto el sitio de la conferencia como el sitio de la cadena de hoteles publican las direcciones con un modelo de datos que pueda ser procesado por otra aplicación web. En este caso la aplicación de mapas para realizar el cálculo de las distancias. De esta forma la aplicación de mapas toma la información de otras páginas web y calcula los resultados deseados.

En el ejemplo del planeta 'Plutón' la información es representada con un modelo de datos a partir del cual se generan las diferentes representaciones en diferentes páginas web. Cuando se efectúa la modificación de la información se actualizan todas las página de forma automática.

El lenguaje de datos que utiliza la ***Web Semántica*** para representar la información de forma distribuida es llamado ***Resource Description Framework (RDF)***. En la ***Web Semántica*** se tiene una representación de los datos que describen conceptos reales, a partir de esos datos se genera la representación para ser visualizados por los humanos. Al contar con un modelo de datos procesable por las computadoras, se puede hacer uso de la información dentro del contenido de las páginas web. Las aplicaciones **Web** pueden comprender y utilizar la información disponible.

### 3. Tecnologías de la Web Semántica.

#### *Resource Description Framework (RDF)*

El lenguaje ***RDF (Resource Description Framework)*** es un lenguaje formal para describir información estructurada. El objetivo de este lenguaje es lograr el intercambio de información en la ***Web*** manteniendo el significado original de la misma. El lenguaje **RDF** tiene como objetivo la descripción de información estructurada. Es posible notar la diferencia en los objetivos de lenguajes como **HTML** o **XML** los cuales se centran en la forma en que se muestran los documentos. Es decir, su objetivo es describir la presentación de los datos. La ventaja de **RDF** es que permite un posterior procesamiento o composición de la información. Por esta razón **RDF** es habitualmente visto como el formato de representación básico para el desarrollo de la ***Web Semántica***.

Un documento **RDF** es la descripción de un grafo dirigido. Dentro de un grafo, existe un conjunto de nodos y aristas direccionadas. Cada nodo está debidamente rotulados para ser distinguidos. La estructura de

grafo se debe a que **RDF** está diseñado para la descripción de relaciones generales entre los objetos de interés. En **RDF** estos objetos son habitualmente llamados *recursos*. Estas relaciones no representan necesariamente una estructura jerárquica, sino que con las sucesivas relaciones de los *recursos* se está describiendo el dominio de interés y de forma natural se forma la estructura de grafo.

Utilizar una estructura de grafo permite el fácil almacenamiento de forma distribuida que es una práctica común en la ***Web.*** Otra importante ventaja es la posibilidad de *composición* que tienen los grafos. Con la estructura de grafo es fácil realizar la *unión o composición* de información proveniente de diversas fuentes. La ventaja radica en que un grafo con menos números de aristas o nodos, también forma un grafo. La facilidad que proveen los grafos es estructural. Realizar operaciones de composición de información y seguir manteniendo la estructura es sencillo y escalable.

Un problema esencial en esta tarea es la unificación de identificadores. Aunque dos documentos hablen del mismo tema, es posible que tengan diferentes identificadores. Por un lado a un mismo *recurso* se le pueden dar diferentes identificadores, o un mismo identificador puede ser utilizado para dos *recursos* diferentes. Esto se debe a que no hay un acuerdo previo sobre la identificación de *recursos*.

Para solucionar el problema de la identificación, **RDF** utiliza *Uniform Resource Identifiers (URIs)* como mecanismo de identificación de los recursos. Así se obtiene una forma clara de distinguir diferentes recursos entre sí. Un identificador *URI* es también un *URL (Uniform Resource Locators)* válido. Todo *URL* válido también es un *URI* válido, y los *URLs* de hecho pueden ser utilizados como identificadores en **RDF**.

Las direcciones son utilizadas con el fin de identificar univocamente a los recursos. El recurso no necesariamente debe estar en esa dirección *URL*. La utilización de este estándar es solamente en lo referido a la identificación. Con lo cual no siempre el objetivo es intercambiar información sobre direcciones de páginas ***Web***. El mecanismo se utiliza para la identificación de todo tipo de recursos abstractos y la relación entre los mismos. Por ejemplo, en el contexto de cierta aplicación se puede querer identificar libros, lugares, personas, eventos, etc. Las *URIs* que no son *URLs* son a veces llamadas *Uniform Resource Names (URNs)*.

La representación de dominios muchas veces requiere la representación de información de valores concretos como números, fechas o valores de verdad. Es deseable que se haga un mínimo manejo entendiendo el significado de los valores. Estos datos son representados por los llamados *literales*. Los literales son nombres reservados para los recursos en **RDF**. El valor de cada literal es generalmente descripto por una secuencia de caracteres, por ejemplo '04' o '59'. Conocer el tipo de datos es crucial para entender su significado, por ejemplo la cadena de caracteres '42' y '042' son los mismos números naturales, pero diferentes cadena de caracteres o *strings*.

Cuando se está describiendo un dominio de interés se introducen nuevos términos. Se lo hace para los individuos y también para sus relaciones. Un individuo puede ser una persona o una universidad que como veremos son clases y una relación puede ser 'empleado de' o 'estudia en'. Cuando se introduce este vocabulario, el usuario va a tener de forma natural una idea concreta del significado de los términos utilizados. Pero desde el punto de vista de sistemas de computadoras, siguen siendo cadenas de caracteres sin ningún

significado definido. Por lo cual, las relaciones tienen que ser comunicadas de forma explícita al sistema en algún formato que permita sacar conclusiones. Para realizar la especificación esa información se utiliza **RDF Schema (RDFS).** La información que se especifica con **RDFS** es llamada *conocimiento terminológico* o *esquema de conocimiento*. En primer lugar se debe decir que **RDFS** no es nada mas que un vocabulario de un **RDF** en particular. Por consiguiente, todo documento **RDFS** es un documento **RDF** bien formado. Esto asegura que los documentos **RDFS** pueden ser procesados por cualquier herramienta compatible con **RDF**.

**RDFS** no agrega ningún vocabulario de un dominio en particular, la intención de **RDFS** es proporcionar construcciones genéricas de lenguaje. Con estas construcciones, un vocabulario definido por un usuario puede ser caracterizado semánticamente. Esto permite la definición de un nuevo vocabulario especificando su 'significado' en el documento sin la necesidad de cambiar la lógica del programa de procesamiento. En resumen, un documento **RDFS** es una especificación procesable por un sistema, que describe el conocimiento acerca de algún dominio de interés. Una de las funcionalidades básicas que debe definir una especificación formal de conocimiento es el concepto de '*tipo*'. En **RDF** esto puede ser mediante el uso de rdf:type. La **URI** rdf:type indica que un recurso es una *instancia* de una *clase*.

Si se desea indicar que un libro es un libro de textos, se lo puede hacer afirmación:

```
libro:uri          rdf:type          ej:LibroTexto
```

Se indica que libro:uri es miembro de la clase de todos los libros de texto. Como se ve en el ejemplo, es claro que se pueden definir nuevas clases definidas por el usuario.

También se puede ver en el ejemplo, que no es posible deducir sintácticamente si LibroTexto es un objeto único o una clase. Por lo cual, **RDFS** provee la forma de indicar que LibroTexto es en efecto una clase. Esto se hace indicando el *tipo* de LibroTexto:

```
ej:LibroTexto      rdf:type          rdfs:Class
```

Se deduce que en rdfs:Class están incluidas todas las clases. Si bien no se puede deducir que ej:LibroTexto es una clase, se utiliza la convención de que todas las clases comienzan capitalizadas.

Se presenta un ejemplo de archivo en **RDF**:

```
#filename: ex001.ttl
@prefix ab: <http://learningrdf.com/ns/addressbook#> .

ab:richard        ab:homeTel   '(229) 276 - 5135' .
ab:richard        ab:email     'richard49@hotmail.com' .

ab:cindy          ab:homeTel   '(194) 966-1505' .
ab:cindy          ab:email     'cindy@gmail.com' .

ab:craig          ab:homeTel   '(194) 966-1505' .
ab:craig          ab:email     'craigellis@yahoo.com'  .
ab:craig          ab:email     'c.ellis@usairwaysgroup.com' .
```

En primer lugar, se indica cual es el nombre del archivo que contiene los datos, luego se define cual será el prefijo que se utiliza. A partir de ahí se comienza a describir los datos representados. Cada línea de descripción de datos es una afirmación que se realiza. En cada linea se describe información en términos de un objeto que se describe un predicado que indica qué es lo que se indica y un sujeto o valor que indica que valor toma el objeto para el predicado indicado. Por ejemplo, la primer línea se puede leer como 'El teléfono particular (predicado) de Richard (objeto) es el (229) 276 – 5135 (valor)' .

**RDF(S)** es adecuado para modelar *ontologías* simples y para la derivación de conocimiento implícito. **RDF** provee solo un conjunto limitado de medios de expresión. Ciertas construcciones de conocimiento complejas no son posible de representar. Por ejemplo oraciones como '*Cada uno de los proyectos tiene al menos un participante*' o '*Los proyectos son siempre internos o externos*'.

Para esto comúnmente se utilizan lenguajes basados en lógica formal. Estos lenguajes nos permiten también hacer razonamiento lógico del conocimiento y acceder al conocimiento que se está modelando de forma implícita.

***Web Ontology Languaje (OWL).***

El objetivo de **OWL** es tener una balance entre la expresividad y la eficiencia de razonamiento. Se responde a la necesidad de contar con diferentes niveles de expresividad, por lo cual se brinda tres sublenguajes, llamados *especies* de **OWL**: **OWL Full**, **OWL DL** y **OWL Lite**. **OWL Full** contiene **OWL DL** y **OWL Lite**, y **OWL DL** contiene **OWL Lite**.

Los documentos **OWL** son utilizados para el modelado de las *ontologías* **OWL**. Para su definición dos tipos de sintaxis han sido estandarizados. Una de las sintaxis está basada en **RDF** y es usualmente utilizada para el intercambio de información. También es llamada sintaxis **OWL RDF** ya que también son documentos **RDF** válidos. La otra sintaxis se llama *sintaxis abstracta* **OWL** y de alguna forma es mas legible para los humanos. Se

utiliza con mayor frecuencia la sintaxis **RDF**.

Una *ontología* en **OWL** esta básicamente expresada en término de clases y propiedades. Estos conceptos son tomados de **RDF**, pero en **OWL** es posible describir relaciones mas complejas entre clases y propiedades. En el encabezado de los documentos **OWL** se da información acerca de los espacios de nombres, versiones, y las anotaciones. Esta información no tiene impacto directo en el conocimiento expresado por la *ontología*. También se pueden definir *individuos* que son instancias de clases. Las propiedades en **OWL** también pueden ser llamadas roles.

La definición de una clase:

```
<rdf:Descripcion rdf:about='Profesor'>
      <rdf:type rdf:resource='&owl;Class'/>
</rdf:Descripcion>
```

Como en **RDF**, los individuos pueden ser declarados como instancias de clases. A eso se le llama asignación de clase.

```
<rdf:Descripcion rdf:about='alan'>
      <rdf:type rdf:resource='Profesor'/>
</rdf:Descripcion>
```

En **OWL** hay dos tipos de roles: los roles abstractos y los roles concretos. Los roles abstractos conectan individuos con individuos. Los roles concretos conectan individuos con valores de datos, por ejemplo elementos de tipos de datos. Los roles se declaran de forma similar a las clases:

```
<owl:ObjectoProperty    rdf:about='tieneAfiliacion' />
<owl:DatatypeProperty   rdf:about='primerNombre' />
```

El primero es un rol abstracto y expresa que organización tiene afiliación. El segundo rol es concreto y asigna el primer nombre a una persona. El dominio y el rango pueden ser declarados con rdfs:domain y rdfs:range.

Las clases **OWL** pueden ser relacionadas por medio de la relación rdfs:subClassOf. Un ejemplo es:

```
<owl:Class rdf:about='Profesor'>
      <rdfs:subClassOf  rdf:resource='MiembroFacultad'      />
</owl:Class>
```

Existen tres sublenguajes de **OWL**. Los diferentes sublenguajes tienen diferencias conceptuales y se los denomina: **OWL FULL**; **OWL DL**; **OWL Lite**.

En **OWL Full** se pueden utilizar todos los constructores de **OWL**. Se los puede combinar libremente con todos los constructores del lenguaje **RDF(S)** mientras el documento resultante sea un documento **RDF(S)** válido. Dada esta falta de restricción en la utilización de constructores de **OWL** y **RDF(S)** se originan varios problemas. Esto sugiere la necesidad de diseñar sublenguajes más restrictos.

Los problemas son causados por el hecho de que **OWL** fue diseñado para modelar conocimiento de

forma expresiva, y la habilidad para acceder al conocimiento de forma implícita utilizando la inferencia lógica. Sin embargo las inferencias en **OWL Full** son indecidibles y en realidad no existe una herramienta de software que soporte toda la semántica de **OWL Full**.

En el sublenguaje **OWL DL** el uso de algunos elementos del lenguaje **OWL Full** están restrictos. Los elementos que están restrictos son los de separación de tipos y utilizar constructores del lenguaje **RDF(S)**. La motivación en el diseño fue lograr que **OWL DL** sea decidible.

Se diseño **OWL Lite** con la intención de ser fácil de implementar un sublenguaje conteniendo los constructores de lenguaje más importantes. Sin embargo, resultó que **OWL Lite** es esencialmente tan dificultoso de trabajar como lo es **OWL DL**. Por lo cuál no es sorpresa que juega un papel menor en la práctica.

***SPARQL.***

Se presenta el lenguaje de consultas para modelos de datos con ***RDF***. El nombre proviene de las siglas definidas de forma recursiva de ***SPARQL Protocol and RDF Query Language*** siendo descripto por un conjunto de especificaciones de la **W3C** [9]. El modelo de datos de **RDF** establece que los datos se describen como triplas, contando con un sujeto, un predicado y un objeto. Cada tripla es como una afirmación que establece un hecho o conocimiento. Por ejemplo podemos tener las siguientes afirmaciones:

| Sujeto | Predicado | Objeto |
|---|---|---|
| ricardo | telParticular | (221) 452 – 2012 |
| clara | email | clara@gmail.com |

Tabla 1. Ejemplos de triplas.

En la *tabla 1* se puede ver que el sujeto es el recurso identificado, el predicado el nombre de la propiedad y el objeto el valor de la propiedad. La primer tripla se la puede interpretar como una afirmación que indica lo siguiente: *'el teléfono particular de ricardo es el (221) 452 – 2012'.*

Una consulta en **SPARQL** debe ser entendida como 'Quiero esta parte de la información del subconjunto de datos que cumple con estas condiciones'. Las condiciones se describen con patrones de triplas, que son similares a las triplas **RDF**, pero que pueden incluir variables. Las variables agregan flexibilidad al momento de comparar los patrones con la información.

A modo de ejemplo se presenta un archivo de una consulta. La consulta se realizara con los datos del archivo '*ex001.ttl*' presentado en la sección de **RDF**.

```
#filename: ex002.rq
PREFIX      ab: <http://learningsparql.com/ns/addressbook#>
SELECT            ?craigEmail
WHERE
{     ab:craig          ab:email    ?craigEmail . }
```

En esta consulta se pretende encontrar 'la dirección de email asociada con craig'. Las palabras PREFIX, SELECT y WHERE son puestas en mayúsculas solo por convención. Por lo cual, esta consulta nos retorna todas las triplas que cumplan con la condición.

En la siguiente imagen vemos como funciona gráficamente una consulta en **SPARQL**.

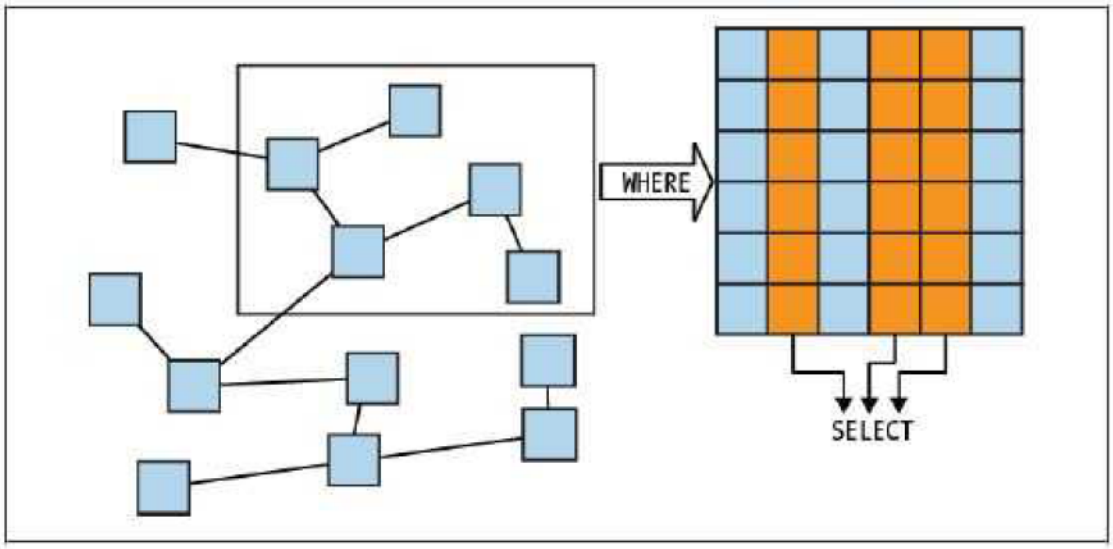


Imagen 8. Esquema de una consulta en **SPARQL**.

La palabra clave WHERE indica cuales son la triplas que se deben seleccionar. Luego, con SELECT seleccionamos que parte de la información que cumple con las condiciones deseamos obtener. En el ejemplo anterior, de todas las triplas que cumplen con las condiciones dentro del WHERE solamente deseamos obtener la parte de la tripla que esta asociada con la variable ?craigEmail.

El resultado que se obtiene de ejecutar la consulta **SPARQL** con los datos del archivo '*ex001.ttl*' son:

| craigEmail |
|---|
| 'c.ellis@usairwaysgroup.com' |
| 'craigellis@yahoo.com' |

Tabla 2. Resultados de la consulta.

En la consulta anterior vemos que la variable esta en la posición de predicado de la tripla. Es posible incluir dentro de una condición mas de una variable. A continuación se muestra una consulta con dos variables:

```
#filename:  ex0010.rq
PREFIX      ab:   <http://learningsparql.com/ns/addressbook#>
SELECT      ?propertyName     ?propertyValue
WHERE
{ab:cindy   ?propertyName     ?propertyValue    . }
```

Con esta consulta estamos solicitando toda la información asociada con ab:cindy. De forma que nos retorna el nombre de la propiedad con el valor que contiene. Se muestran los valores obtenidos:

| propertyName | propertyValue |
|---|---|
| ab:email | 'cindy@gmail.com' |
| ab:homeTel | '(245)  646-5488' |

Tabla 3. Resultados.

La consulta nos retorna todos las propiedades asociadas con ab:cindy con lo cual obtenemos la propiedad ab:email con su valor y la propiedad ab:homeTel con su correspondiente valor.

Puede suceder que necesitemos realizar una búsqueda de un texto en particular. La siguiente consulta devuelve todas las tuplas que contenga un texto determinado:

```
#filename:  ex21.rq
PREFIX      ab:   <http://learningsparql.com/ns/addressbook#>
SELECT      *
WHERE
{     ?s    ?p    ?o .
      FILTER      (regex      (?o, 'yahoo', 'i'))
}
```

En la consulta se busca por todas las triplas que contengan el texto 'yahoo' en la parte del objeto. Por convención cuando se busca triplas sin nombres de propiedad (variables en las tres posiciones) se utilizan las variables ?s (subject), ?p (predicate) y ?o (object). La función *regex ()* realiza una comparación con un patrón de búsqueda y la variable que se indica. De esta forma si el patrón de búsqueda se encuentra en la variable retorna *true* y *false* en caso contrario. La palabra clave FILTER realiza un filtrado de las tuplas que cumplen con la condición que contiene. En esta caso se selecciona solamente aquellas triplas para las cuales *regex()* retorna *true*.

La función retorna *true* si el texto se encuentra en la variable que se pasa por parámetro. Otra particularidad de esta consulta es que junto a la palabra clave SELECT se ve un '*****' esto indica que queremos visualizar todos los campos de las triplas que retorna la consulta.

Muchas veces resulta tedioso escribir una consulta en **SPARQL**. Por este motivo se da la posibilidad de una sintaxis mas compacta.

```
#filename:  ex22.rq
PREFIX      ab:   <http://learningsparql.com/ns/addressbook#>
SELECT            ?first            ?last
WHERE
      {
      ?person     ab:homeTel        '(229)276-5135';
                  ab:firstName      ?first;
                  ab:lastName       ?last.
      }
```

En la consulta anterior estamos consultando por el primer nombre y el apellido de la persona que tiene el numero de teléfono particular '(229) 276-5135'. Para no tener que repetir '?person' en cada una de las triplas de la condición, en lugar de terminar la primer línea con un punto, se termina con un punto y coma. De esta manera se estamos diciendo a **SPAQRL** que el sujeto de la segunda tripla es el mismo que el de la anterior.

Puede suceder que necesitemos incluir una propiedad dentro de las condiciones. Para que una tripla sea devuelta tiene que cumplir con todas las condiciones de la clausula WHERE. Lo que sucede es que muchas veces algunas cumplen con ciertas condiciones y con otras no.

```
#filename:  ex23.rq
PREFIX      ab:   <http://learningsparql.com/ns/addressbook#>
SELECT            ?first      ?last ?workTel    ?nick
WHERE
{
      ?s    ab:firstName      ?first      ;
            ab:lastName       ?last .
      OPTIONAL {  ?s    ab:workTel  ?workTel.   }
      OPTIONAL {  ?s    ab:nick     ?nick.            }
}
```

En la consulta anterior se retornan todas aquellas tuplas que cumplan con la condición y que opcionalmente tengan 'ab:workTel' y 'ab:nick'. Al estar dentro de la clausula OPTIONAL aquellas triplas que no tengan 'ab:work' y 'ab:nick' también son incluidas en el resultado de la consulta. Se indica en dos bloques

debido a que se requiere que aquellas triplas que tengan, ya sea ab:workTel ó ab:nick se muestren los valores de uno de los dos. Si se pusieran dentro de un solo bloque OPTIONAL para que se muestren los valores deben tener ambos valores. Es decir, si una tripla no cumple con la condición dentro del bloque OPTIONAL es incorporada a la respuesta ahora si es un bloque solo y la tripla tiene un solo valor, no se muestra ninguno de los dos. En cambio, si ponemos dos bloques separados si al menos tiene uno de los dos valores, el valor será mostrado.

En una base de datos relacional cada línea de cada tabla necesita un identificador único. Este identificador es el que se utiliza para hacer un referencia a esa línea en otra tabla. En ***RDF*** cada una de las triplas cuenta con un identificador único. Por esta razón en ***RDF*** es mas sencillo realizar '*joins*' o '*union*' de mas de una tabla con referencias cruzadas. En **SPARQL** la versión del '*join*' se realiza de la siguiente manera:

```
#filename:  ex070.rq
PREFIX      ab:   <http://learningsparql.com/ns/addressbook#>
SELECT      ?last ?first      ?courseName
WHERE
{
      ?s          ab:firstName      ?first ;
                  ab:lastName       ?last ;
                  ab:takingCourse   ?course .
      ?course     ab:courseTitle    ?courseName.
}
```

En la consulta anterior, **SPARQL** primero busca asigna a la variable ?course el identificador del curso que el alumno esta tomando. Luego busca por un valor de ab:courseTitle para ese curso que esta tomando el alumno. De esta manera se llega al nombre del curso que esta tomando el alumno y es asignado a la variable ?courseName.

Como sucede en **SQL** en **SPARQL** contamos con una palabra clave para no obtener valores duplicados en la respuesta. Esa palabra clave en **SPARQL** es DISTINCT. Muchas veces es necesario obtener solo resultados únicos, para esto luego de la palabra clave SELECT introducimos la palabra clave DISTINCT y de esta manera le decimos a **SPARQL** que solo queremos resultados únicos.

Puede suceder que realicemos una consulta para la cual obtenemos un gran numero de respuestas. Por lo cual es necesario poder establecer cual es el limite máximo de respuestas que queremos. Para esta tarea **SPARQL** cuenta con la palabra clave LIMIT. Agregando al final de la consulta esta palabra clave con un valor numérico establecemos la cantidad máxima de triplas que obtenemos como resultado, sin importar la cantidad de triplas que responden al patrón que escribimos en la consulta. Por ejemplo, vemos una consulta que sin la palabra LIMIT obtendríamos muchas respuestas:

```
#filename:  ex114.rq
SELECT      ?label
WHERE
{ ?s  rdfs:label  ?label.     }
LIMIT 10
```

Si realizamos esta consulta sin LIMIT en Dbpedia [10] retornaría millones de resultados, de esta manera limitamos el resultado a 10 triplas.

La palabra clave OFFSET indica a **SPARQL** que debe ignorar cierta cantidad de triplas y luego comenzar a retornar las triplas como resultado. Por ejemplo, si al final de la consulta ponemos OFFSET 3, se ignoran las primeras triplas que retornaría la consulta sin OFFSET. El resto de las triplas son retornadas como es habitual. Es posible realizar una combinación de OFFSET y LIMIT, lo cual indica que primero se ignoren la cantidad de triplas que indica OFFSET y luego se retorna la cantidad de triplas que indica LIMIT.

Una vez que **SPARQL** obtiene valores del conjunto de datos, se pueden realizar operaciones o llamar funciones con esos datos. Se muestra un ejemplo:

```
#filename: ex139.rq
PREFIX      e:<http://learningsparql.com/ns/expenses#>
SELECT      ?description      ?amount     ((?amount * .2 ) AS ?tip)
      ((?amount + ?tip) AS ?total )
WHERE
{
      ?meal e:description     ?description;
            e:amount          ?amount.
}
```

En la consulta anterior obtenemos la descripción y el monto de una comida, luego calculamos la propina y calculamos cual es el total de la comida. Las información obtenida sería la siguiente:

| description | amount | tip | total |
|---|---|---|---|
| 'dinner' | 28,3 | 5,66 | 33,96 |
| 'lunch' | 11,13 | 2,22 | 13,35 |
| 'breakfast | 5,53 | 1,3 | 7,83 |

Tabla 4. Resultados

Una nueva incorporación es la palabra AS que indica como se debe llamar la nueva columna que

calculamos a partir de los valores de otras columnas. Ademas de realizar operaciones matemáticas podemos realizar llamados a funciones matemáticas.

En **SPARQL** es posible ordenar los resultados que obtenemos. Incorporando la palabra clave ORDER BY a final de la consulta indicamos el campo por el cual nos interesa ordenar los datos resultantes. Por defecto se ordenan de forma ascendente. Luego si nos interesa ordenar los datos de forma descendente, solo tenemos que utilizar la función DESC(). También es posible ordenar por mas de un valor. A continuación se muestra un ejemplo:

```
#filename:  ex146.rq
PREFIX      e:    <http://learningsparql.com/ns/expenses#>
SELECT            ?description      ?date ?amount
WHERE
{
      ?meal e:description     ?description ;
            e:date            ?date ;
            e:amount    ?amount .
}
ORDER BY ?description   DESC(?amount)
```

En la consulta anterior los datos son ordenados de la siguiente manera:

| description | date | amount |
|---|---|---|
| 'breakfast' | '2001-10-16T09:05' | 6,65 |
| 'breakfast' | '2001-10-14T012:05' | 6,53 |
| 'breakfast' | '2001-10-15T08:05' | 4,32 |
| 'dinner' | '2001-10-15T08:05' | 31,45 |
| 'dinner' | '2001-10-14T08:05' | 28,30 |
| 'dinner' | '2001-10-16T19:05' | 25,05 |
| 'lunch' | '2001-10-14T19:05' | 11,13 |
| 'lunch' | '2001-10-16T08:05' | 10,00 |
| 'lunch' | '2001-10-15T12:05' | 9,45 |

Tabla 5. Resultados.

Como vemos los datos primero se ordenan por la descripción, luego entre las triplas de una misma descripción se ordenan de forma descendente de acuerdo al valor (*'amount'*).

Otra característica de **SPARQL** es la capacidad de agrupar conjuntos de datos para calcular por ejemplo un subtotal del conjunto. Por ejemplo, podemos modificar la ultima consulta para obtener los valores sumados para todos los desayunos ('*breakfast')* para las cenas ('*dinner*') y almuerzos ('*lunch*').

```
#filename:  ex160.rq
PREFIX      e:    <http://learningsparql.com/ns/expenses#>
SELECT      ?description      (SUM(?amount) AS ?mealTotal)
WHERE
{
      ?meal e:description     ?description ;
            e:amount          ?amount .
}
GROUP BY    ?description
```

Esta consulta nos retorna los siguientes datos:

| description | meatTotal |
|---|---|
| 'dinner' | 84,8 |
| 'lunch' | 30,58 |
| 'breakfast' | 17,5 |

Tabla 6. Resultados.

En el ejemplo anterior se puede sustituir SUM, que suma los valores de cada subconjunto por otras funciones como AVG() que realiza el promedio de los valores, MIN() que retorna el mínimo de los valores, MAX() que retorna el máximo valor, o COUNT() que retorna la cantidad de valores que tiene cada subconjunto.

La palabra clave HAVING realiza para GROUP BY la misma tarea que realiza FILTER para valores individuales. Se utiliza para especificar una condición que debe cumplir el conjunto para formar parte del resultado.  A continuación se muestra un ejemplo de utilización:

```
#filename:  ex164.rq
PREFIX      e:    <http://learningsparql.com/ns/expenses#>
SELECT      ?description      (SUM(?amount) AS ?mealTotal)
WHERE
{
      ?meal e:description     ?description ;
            e:amount          ?amount .
}
GROUP BY    ?description
HAVING (SUM(?amount) > 20 )
```

La condición de HAVING indica que la suma de ?amount debe ser mayor que 20 para formar parte del resultado de la consulta. De esta forma se puede realizar un filtrado de subconjuntos que deben pertenecer a la respuesta de la consulta.

El lenguaje de consulta **SPARQL** propone describir grafos de información que deben coincidir con aquellas triplas que se deben retornar. Dentro de los grafos se pueden utilizar variables, para describir relaciones dentro de la información necesaria y diferentes formas de filtrar la información que deseamos obtener.

### 4. Wikis Semánticas.

Una ***Wiki Semántica*** combina las propiedades de las ***Wikis*** tradicionales con las tecnologías de la ***Web Semántica***. Se mantiene las propiedades de las ***Wikis*** como la libertad y flexibilidad de uso, el ambiente colaborativo y la capacidad de realizar *links* entre *artículos wikis*. A estas propiedades se le agregan las tecnologías de la ***Web Semántica*** como el contenido estructurado, modelos de conocimiento en forma de *ontologías* y búsquedas *semánticas*.

Las ***Wikis*** tradicionales tuvieron éxito gracias a la simplicidad con la que se presentan a los usuarios. Dado que las ***Wikis Semánticas*** continúan siendo fáciles de utilizar, son un buen punto de partida para la introducción de las nuevas tecnologías de la ***Web Semántica*** que en otras aplicaciones pueden resultar difíciles de comprender. Para realizar una asociación entre una estructura y el contenido de un *artículo wiki*, los usuarios agregan anotaciones semánticas al contenido. Por ejemplo, se agregan anotaciones a los *links* con lo cual resulta sencillo consultas del tipo *'todo lo que esta linkeado o relacionado aquí'*.

En una ***wiki semántica*** los usuarios generan el contenido junto con las anotaciones semánticas. Para los usuarios agregar anotaciones semánticas es un trabajo adicional. El beneficio es que luego estas anotaciones les permiten realizar consultas semánticas, y relacionar la información de forma más sencilla.

Dentro del término ***wiki semántica*** se agrupa una variedad de aplicaciones. La representación interna de las anotaciones en ***RDF/OWL*** simplifica el intercambio de datos con otras aplicaciones. Por ejemplo, se puede proveer búsquedas externas como un servicio web. Adicionalmente los sistemas ***Wikis*** pueden utilizar

razonamiento deductivo para derivar conocimiento adicionales para los usuarios.

Las ***wikis semánticas*** mejoran la experiencia del usuario con nuevas funcionalidades. Se provee un formalismo sencillo para la generación de anotaciones. Tanto dentro del contenido de las ***Wikis*** como en los *links* entre páginas y sobre diferentes tipos de contenido (por ej. una imagen donde se puede definir su significado o su *semántica*).

Una vez que los usuarios generan los contenidos de una ***Wiki***, es interesante la forma acceder a ellos. Mediante las búsquedas semánticas se puede acceder al contenido no solo con palabras claves, sino también con relaciones semánticas entre objetos.

Como sucede con las ***Wikis*** tradicionales, no existe un estándar para una ***Wiki Semántica***. Diferentes sistemas persiguen diferentes objetivos y tienen diferentes restricciones. En el caso de ***Semantic Media Wiki*** [4] se concentra en un escenario de enciclopedia como Wikipedia. Por este motivo es importante la escalabilidad y la compatibilidad con versiones anteriores. Otro ejemplo es ***IkeWiki*** [8] que es una aplicación que originalmente fue desarrollada para la creación de *ontologías* de forma colaborativa y para el manejo de conocimiento. Por esta razón se centra principalmente en proveer funciones avanzadas de semántica como el razonamiento. Esta diferencia de objetivo permite a ***IkeWiki*** aceptar menos escalabilidad y una mayor demanda de hardware que ***Semantic Media Wiki***.

Las ***wikis semánticas*** puede solucionar problemas de acceso a la información. Teniendo las anotaciones semánticas y por lo tanto una estructura, el sistema puede proveer al usuario con herramientas para mejorar la facilidad de uso. Por ejemplo, realizar una correcta visualización para planificaciones de proyectos, intercambiar anotaciones con otras ***Wikis*** o aplicaciones y ofrecer búsquedas semánticas.

Las ***wikis semánticas*** crean una *ontolgía* siendo ésta una tarea compleja de realizar. Los usuarios expertos en ciertos dominios como la biología, conocen en profundidad el dominio, pero no cuentan con los conocimientos para el desarrollo de una *ontología*. Los expertos en informática por otro lado, saben como realizar correctamente una *ontología*, pero carecen de conocimiento del dominio. Las *ontologías* de mejor calidad existen en áreas como la medicina o biología, donde es aceptable realizar un esfuerzo extra para su construcción.

Para la construcción de una *ontología* se requiere de trabajo colaborativo entre diferentes profesionales. Las ***wikis semánticas*** pueden contribuir significativamente en esta tarea. Se puede comenzar por una descripción textual en las *páginas wikis*, realizada por los expertos de dominio. Luego, los expertos en computación pueden interactuar con los expertos del dominio para formalizar el conocimiento. Otros usuarios opcionalmente, podrán refinar la *ontología* utilizando diferentes herramientas.

### 5. Semantic Media Wiki.

**Semantic Media Wiki (SMW)** [5] es una de las ***wikis semánticas*** más utilizadas debido a su facilidad de uso. Se pretende lograr que las tecnologías semánticas estén disponibles para una amplia comunidad. Para cumplir con este objetivo se integran las tecnologías semánticas con la utilización habitual de **Media Wiki,** siendo la ***Wiki detrás*** de Wikipedia. El objetivo final de **SMW** es soportar '*Semantic Wikipedia*', por esta razón se debe tener en cuenta la usabilidad y la escalabilidad tanto como la capacidad de las características semánticas.

***Semantic Media Wiki*** es un motor ***wiki*** mejorado semánticamente que posibilita a los usuarios utilizar anotaciones semánticas en el contexto de la ***Wiki***. Dado el tipo de sistema que es una ***Wiki***, proveer la tecnología para utilizar anotaciones semánticas no es suficiente. La clave radica en proveer tecnologías accesibles para usuarios generales que no son expertos.

**Realizando anotaciones.**

En **SMW** la información semántica se agrega por el usuario incluyendo anotaciones en el contenido de los artículos. Cada artículo pertenece a exactamente un elemento ontológico (incluyendo clases y propiedades). Cada anotación en un artículo realiza una declaración sobre ese elemento. En la siguiente imagen vemos cual es la sintaxis para realizar una anotación:

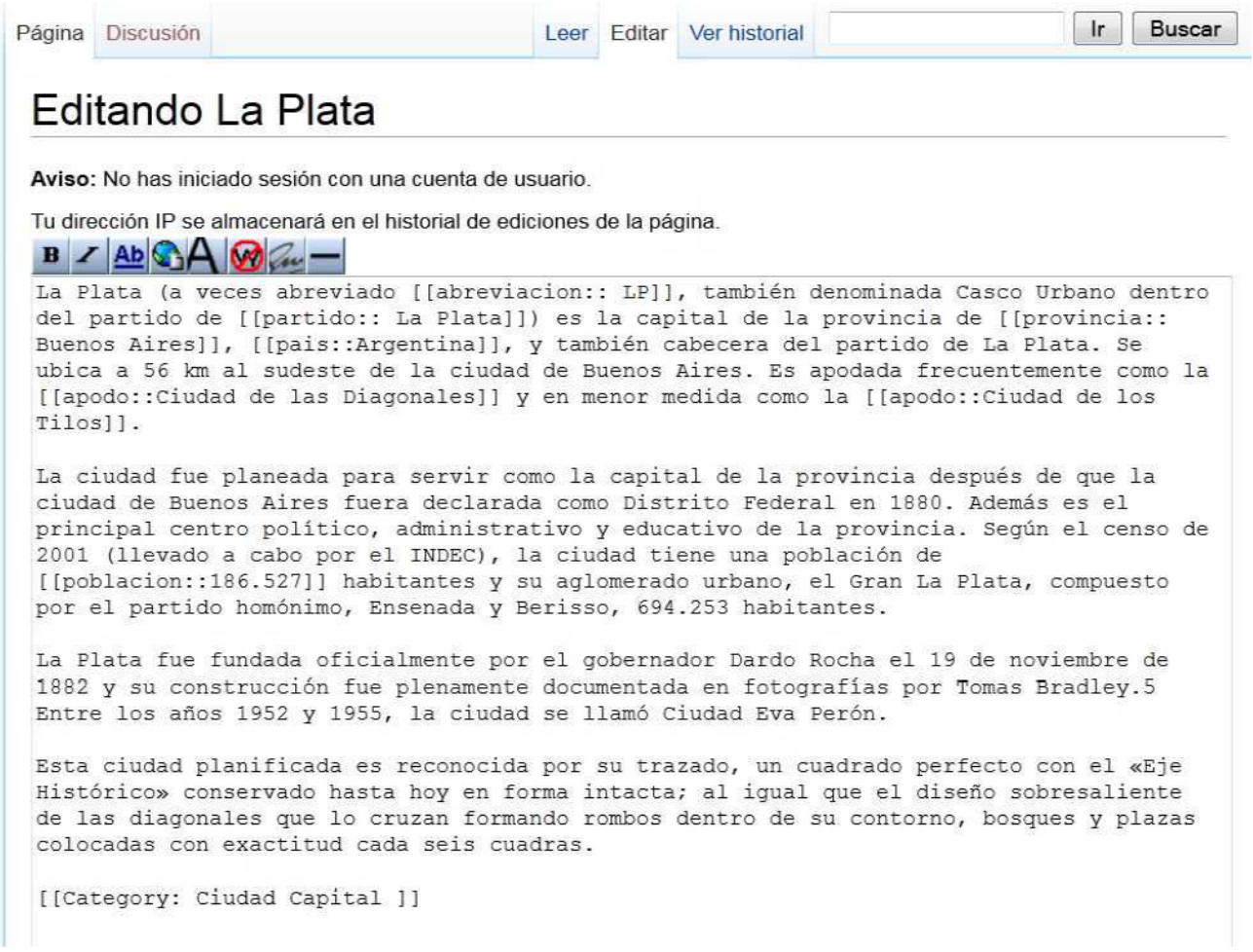


Imagen 9. Realizando anotaciones.

Realizando una anotación en el *artículo wiki* de la ciudad de 'La Plata' realizamos una afirmación sobre la misma. Por ejemplo, la anotación [[ pais :: Argentina ]] indica que la ciudad de 'La Plata' pertenece al país Argentina. Luego, podemos crear otro *artículo wiki* llamado 'Ciudades de Argentina'. En este nuevo artículo podemos escribir una consulta donde se seleccionen todas las ciudades de Argentina. Si por algún motivo decidimos eliminar la anotación [[ pais :: Argentina ]] del *artículo wiki* de la ciudad de 'La Plata', de forma automática se elimina el *link* al artículo 'La Plata" desde el artículo 'Ciudades de Argentina'. En este sentido se dice que una ***Wiki Semántica*** mejora la consistencia de la información. En una ***Wiki*** tradicional se debería eliminar de forma manual al artículo de 'La Plata' del artículo de 'Ciudades de Argentina'.

Una vez que guardamos el artículo podemos visualizar las anotaciones como *links.* En el caso de las anotaciones como relaciones vamos hacia el *artículo wiki* relacionado. Por ejemplo, vemos un *link* hacia el *artículo wiki* del país Argentina, es una anotación que describe una **relación** entre dos *artículos wiki*. Este tipo de relaciones favorecen la navegabilidad de la ***Wiki.***


Imagen 10. Vista del artículo 'La Plata' con propiedades.

Las **categorías** son una forma sencilla de anotación que permite a los usuarios clasificar páginas. Si bien las categorías ya existían en **Media Wiki**, en **SMW** son utilizadas como clases de ***OWL***. Para definir que un artículo pertenece a una categoría simplemente se debe escribir [[Category : NombreCategoria]] dentro del artículo que estamos categorizando. En la imagen 9 vemos como se realiza la anotación, esta anotación puede estar ubicada en cualquier párrafo del texto. Luego en la imagen 10 vemos al pie del artículos cual es la categoría a la que pertenece.

Los **atributos** permiten a los usuarios especificar relaciones entre artículos y objetos que no son

artículos. Realizando la anotación se establece un valor para una propiedad del artículo. Por ejemplo, en el artículo de las imágenes anteriores se puede ver que la propiedad 'poblacion' tiene asociado un valor numérico y no otro *artículo wiki* (no confundir con el hecho que valor numérico tiene automáticamente un artículo que lo representa).

Como se puede ver en la imagen 10, una vez que guardamos el artículo no vemos el nombre de la anotación, sino que se muestra el valor en forma de *link*. Si accedemos a la herramienta 'Explorar Propiedades' podemos ver cuales son todas las propiedades que son utilizadas en el artículo. En la siguiente imagen vemos cuales son las propiedades que son utilizadas en el artículo que describe la ciudad de 'La Plata'.


Imagen 11. Propiedades de un artículo.

Para interpretar los atributos se requiere información acerca del tipo de anotación. Números enteros, cadenas de caracteres y fechas todos requieren una forma de interpretarse diferente. Para eso se debe poder definir el tipo de los atributos. Como se menciono anteriormente, cada elemento ontológico tiene su artículo que lo representa. Existe un artículo por cada categoría, relación y atributo. Esto tiene como ventaja que los usuarios pueden escribir la documentación para cada elemento del vocabulario. Esta documentación resulta crucial para el uso consistente de las anotaciones. Para asignar un tipo a una propiedad se debe declarar en el artículo de la propiedad [[ Type : NombreTipo ]]. Realizando esta declaración se puede interpretar el atributo de forma adecuada.

**Consultas semánticas.**

Una vez que los usuarios generan los contenidos de una ***Wiki*** es interesante la forma acceder a ellos. Las búsquedas semánticas son el mejor medio para acceder al contenido en **SMW**. Los usuarios pueden realizar

búsquedas semánticas de artículos utilizando un lenguaje de búsqueda sencillo. La sintaxis para las consultas semánticas es la misma que se utiliza para realizar anotaciones.

Por ejemplo, si queremos buscar todas las ciudades de Argentina que sean capital y que tengan una población mayor a 100.000 habitantes, debemos escribir la siguiente consulta:

[[category : Ciudad Capital]] [[pais :: Argentina]] [[ poblacion ::> 100 000]]

De esta forma indicamos que tiene que ser una ciudad capital, del país Argentina y que tenga una población mayor a 100 000 habitantes.

En **SMW** brinda la posibilidad de realizar consultas embebidas dentro del texto de las páginas de la ***Wiki***. Las consultas son escritas con texto-wiki y son encapsuladas dentro de los *tags* <ask> y </ask> o con el tag {{ #ask [[condicion]] | parametros }}. Mediante esta funcionalidad los artículos obtienen un contenido dinámico.

Los usuarios de la ***Wiki*** pueden escribir sus propias consultas dentro de los *artículos wiki*. Cuando el artículo es visitado, en el lugar donde se escribió la consulta se muestra el resultado de dicha consulta. Los usuarios también tienen la posibilidad de mostrar diferentes propiedades de los resultados obtenidos y modificar la apariencia de los mismos. En la siguiente imagen vemos como se escribe una consulta embebida:

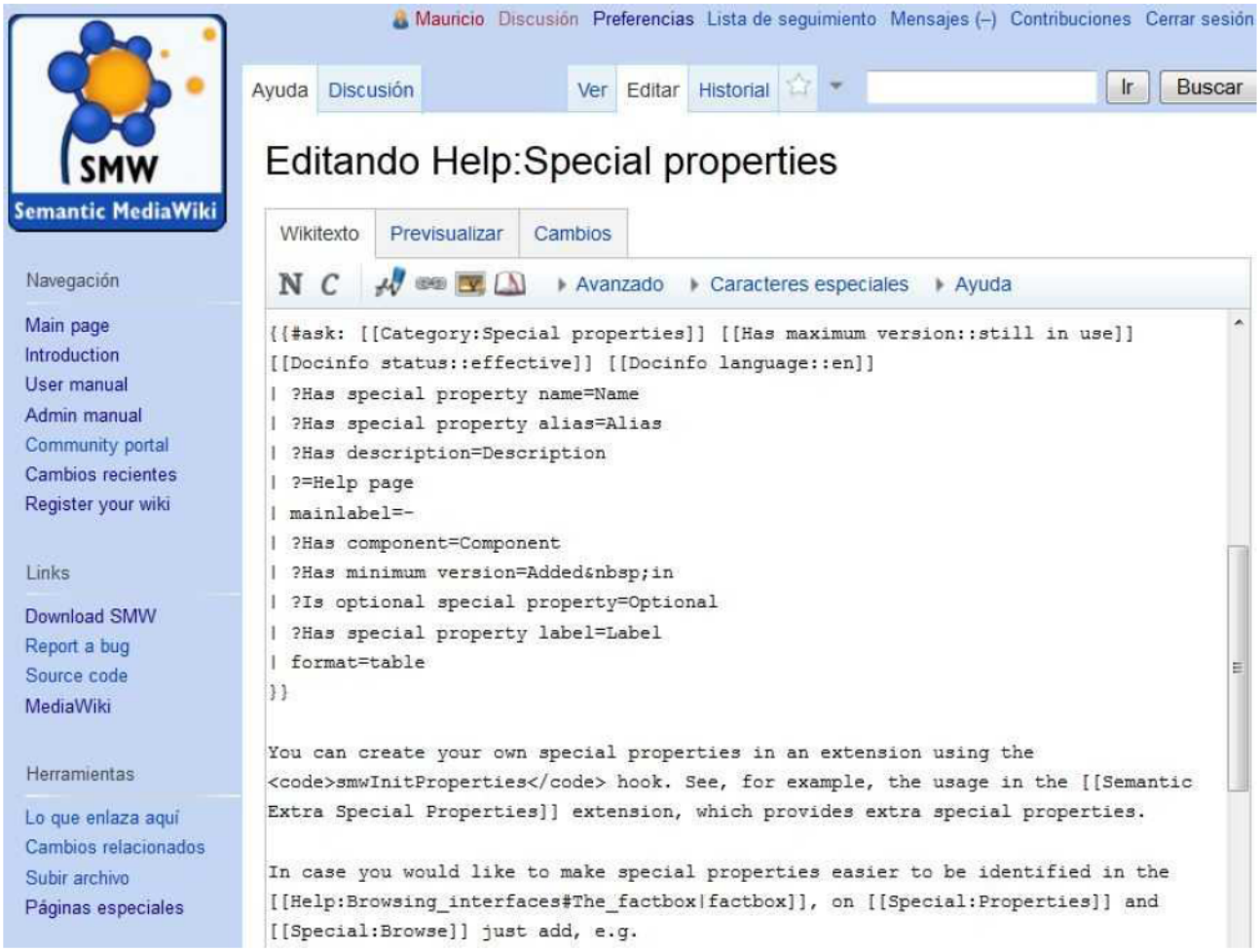


Imagen 12. Escribiendo una consulta embebida.

Utilizando una consulta semántica el artículo obtiene contenido dinámico. Una vez que guardamos el artículo se puede visualizar los artículos que la consulta semántica obtiene. En **SMW** es posible visualizar los resultados con diferentes formatos, en la siguiente imagen vemos los resultados en formato de tabla.

| Name | Alias | Description | Help page | Component | Added in | Optional | Label |
|---|---|---|---|---|---|---|---|
| Allows value | has no alias | Lists one permissible value for a property | Special property Allows value | Semantic MediaWiki | 1.0 | false | not specified |
| Corresponds to | has no alias | Gives the conversion factor for some unit of a physical quantity and the possible names for that unit | Special property Corresponds to | Semantic MediaWiki | 1.0 | false | not specified |
| Creation date | has no alias | It has a fixed value that corresponds to the date of the first revision of each page | Special property Creation date | Semantic MediaWiki | 1.7.0 | true | _CDAT |
| Display units | Display unit | Specifies the a comma-separated list of units or formats that a property should use in display | Special property Display units | Semantic MediaWiki | 1.0 | false | not specified |
| Equivalent URI | has no alias | Marks a page in the wiki as having a well-known meaning beyond this wiki, in an external URI | Special property Equivalent URI | Semantic MediaWiki | 1.0 | false | not specified |
| Has fields | has no alias | Defines a short list of fields with a fixed type and order for datatype Record | Special property Has fields | Semantic MediaWiki | 1.5.0 | false | not specified |
| Has improper value for | has no alias | Alerts in case the assigned value to a property is invalid | Special property Has improper value for | Semantic MediaWiki | 1.4.2 | false | not specified |
| Has subobject | has no alias | Holds the subobjects set on a page | Special property Has subobject | Semantic MediaWiki | 1.7.0 | false | not specified |
| Has type | has no alias | Assigns a type to a property | Special property Has type | Semantic MediaWiki | 1.0 | false | not specified |
| Imported from | has no alias | Allows users to reuse elements of external vocabularies directly within the wiki | Special property Imported from | Semantic MediaWiki | 1.0 | false | not specified |
| Is a new page | has no alias | Marks a page as being new or not | Special property Is a new page | Semantic MediaWiki | 1.7.1 | true | _NEWP |
| Last editor is | has no alias | Holds the page name of the user who created the last page revision | Special property Last editor is | Semantic MediaWiki | 1.7.1 | true | _LEDT |

Imagen 13. Tabla resultado de una consulta.

Junto con los artículos que conforman el resultado podemos ver algunas de sus propiedades. Al escribir la consulta es posible definir cuales propiedades nos interesa saber de los artículos seleccionados. Dentro de los parámetros de la consulta es posible indicar el nombre de cada una de las columnas de propiedades, o el orden en el cuál deseamos visualizar el resultado.

**Reutilización en la Web Semántica.**

Es posible realizar un mapeo de todas las anotaciones de **SMW** a ***OWL***. Como ya mencionamos, cada artículo representa un elemento ontológico. El tipo de los elementos es fijo para la mayoría de las anotaciones. Los artículos normales, son individuos, las categorías se vuelven clases, las relaciones objetos de propiedades entre los artículos. Los atributos pueden ser tipos de datos, anotaciones o objetos de propiedades dependiendo de su tipo. Basado en ese mapeo, **SMW** puede generar para cada página un archivo de exportación en *OWL/RDF*.

Dado que **SMW** es compatible con *OWL DL* es posible reutilizar las ontologías existentes. Existen dos posibilidades para esto, la primera es importando la *ontología*. Mediante esta funcionalidad **SMW** permite crear o modificar páginas en la ***Wiki*** para representar la relación existente en algún documento *OWL DL*. La segunda forma, es la reutilización del vocabulario que permite a los usuarios de la ***Wiki*** mapear las páginas a elementos existentes en la *ontología*.

Para realizar la importación de *ontología* se utiliza la API RAP [ www.wiwiss.fu-berlin.de/suhl/bizer/rdfapi/ ]. Se toma el documento *RDF* y se extrae las declaraciones que pueden ser representadas en la ***Wiki***. Los nombres de los artículos de los elementos importados son derivados de sus etiquetas, o si no existen las etiquetas, de la parte identificadora de su **URI**. El principal propósito de la importación es la generación de una estructura principal para luego completar la ***Wiki***.

La importación del vocabulario le da al usuario la posibilidad de identificar los elementos de la ***Wiki*** con los elementos de una *ontología* existente. Los usuarios de la ***Wiki*** pueden decidir que páginas deben tener semántica externa. Pero los administradores son los encargados de proveer el conjunto de elementos externos disponibles.

**Aspectos de implementación de SMW.**

Se describe las decisiones de diseño mas importantes de **SMW**, las características del modelo de datos que se utiliza para representar la información semántica. Se define *que es* la información semántica dentro de **SMW**.

Las funciones vistas por el usuarios pueden ser divididas en los siguientes unidades:

- **Entrada de datos** (anotaciones en forma de texto).
- **Inspección de datos y *output*** (páginas especiales).
- **Resultado de consultas y formateo de consultas.**
- **Exportación de datos.**
- **Herramientas de mantenimiento** (*scripts* y páginas especiales utilizadas por los administradores del sitio).

Esta división es básica y algunas características pueden entrar en mas de un grupo. De forma adicional se proveen funcionalidades para la Internacionalización. Cada una de las unidades cuenta con su propio código, pero todas están basadas en un *modelo de datos* en común. El modelo de datos provee una forma unificada de representación interna para el manejo de la información semántica. Este es el centro de la arquitectura de **SMW** donde se realizan las decisiones mas esenciales de arquitectura. Esto especifica la forma en la cual la información semántica está representada en el sistema, básicamente define *qué es* la información semántica dentro de **SMW**.

Los datos son representados por pares de propiedad-valor que están asignados a objetos. Si tenemos la siguiente afirmación '*La Plata (objeto) tiene una población (propiedad) de 694,253 (valor)*' se elabora el esquema con las siguientes condiciones:

- Los objetos descriptos son normalmente *artículos wiki.*
- Las propiedades puede ser utilizadas en cualquier lugar. No pertenecen a un *artículo wiki* en particular, categorías, etc.
- Los valores pueden ser de diferentes tipos (números, fechas, otros *artículos wiki*, … ).
- El tipo de los datos es parte de la identidad del valor (valores de diferentes tipos son diferentes, incluso si están basados en la misma entrada de usuario).
- Los objetos pueden tener cero, uno o muchos valores para una propiedad.

- Un hecho semántico es especificado por completo por su objeto, propiedad y valor. Por ejemplo, no interesa quién especifico el hecho (a diferencia de sistemas de *etiquetado (tagging)* donde los usuarios tienen *tags* individuales para una misma cosa).
- Los hechos ocurren o no ocurren, pero no se pueden producir mas de una vez (nuevamente en contraste con los sistemas de *etiquetado (tagging)* donde se cuenta la cantidad de veces que ocurre para un recurso).
- 'Objeto' es un término general, por lo cual usualmente se utiliza 'sujeto' cuando se quiere enfatizar que algo es un sujeto de una asignación propiedad-valor en un hecho.

Estas ideas se ven reflejadas en el modelo básico de datos. Todos los elementos son representados por objetos PHP siendo subclases de *SMWDataItem*. Los conjuntos de hechos semánticos a su vez están representados por objetos del tipo *SMWSemanticData*.

La clase *SMWDataItem* es la forma de representar todo lo que es sabido acerca de algún dato que puede actuar como valor en una propiedad. Esta clase y sus subclases son el bloque básico de construcción de los elementos en **SMW**. El propósito es proveer una interface unificada para todas las *entidades semánticas* con las que **SMW** trabaja. Por ejemplo, números, fechas, geocoordinadas, artículos wikis, y propiedades.

Los objetos de *SMWDataItem* representan piezas de información muy sencillos. Un *dataitem* es como un tipo primitivo de PHP (por ejemplo, strings o number). La identidad de un *dataitem* esta determinada exclusivamente por su contenido. Las características son:

- **Inmutables**: Una vez creados, un *dataitem* no puede cambiar.
- **Independientes del contexto**: El significado de un *dataitem* está solo basado en su contenido, sin tener en cuenta ninguna información contextual (como información acerca de la propiedad a la cuál está asignado).
- **Forma limitada**: los tipos de *dataitems* (números, URLs, páginas, …) que **SMW** soporta son limitados y fijos. Las extensiones no pueden agregar nuevos tipos de *dataitems* y los programadores solo necesitan manejar una lista fija de posibles tipos de *dataitems*.

Ser inmutables es esenciales para los dataitems para funcionar simplemente como valores. Esto impone una restricción a los programadores, pero también simplifica la programación ya que no hay que estar pendiente de que los dataitems cambien.

Los tipos de *dataitems* corresponden a las subclases de *SMWDataItem*. Por conveniencia cada *dataitem* es también asociado con una constante de PHP llamada '*DIType*'. Por ejemplo, en lugar de utilizar un *if-the-else* anidado con muchos chequeos de *instanceof,* se puede realizar un *switch* con *DIType* para manejar diferentes casos.

Las subclases de *SMWDataItem* son: *SMWDIWikiPage* que representa las páginas wikis,

*SMWDIProperty* que representa propiedades, *SMWDINumber* que representa números, *SMWDIString* que representa cadenas de caracteres no mayores a 256, *SMWDIBlob* que representa cadena de caracteres de cualquier longitud, *SMWDIBoolean* que representa valores de verdad, *SMWDIUri* que representa URI, *SMWDITime* que representa un punto en el tiempo en la historia humana o geológica, *SMWDIGeoCoord* que representa una locación en el planeta tierra, *SMWDIContainer* que representa un conjunto de hechos en **SMW** representado por un objeto *SMWSemanticData*, *SMWDIConcept* que representa la entrada y características de un concepto, *SMWDIError* que representa una lista de errores.

Puede parecer una limitación que solo se puedan representar los tipos de datos que representan las clases anteriores. Por ejemplo, no existe un *dataitem* para la representación de una formula química. Esto no quiere decir que en **SMW** no se pueda representar ese tipo de información. La tarea de interpretar esa información básica como un formula química debe ser tratada en alto nivel. Existe el *dataitem SMWDIContainer* que representa 'valores' que consiste en varios hechos en **SMW** (triplas de sujeto-propiedad-valor), casi todas las formas complejas de datos se puede representar con este formato.

Para representar información un dataitem debe ser combinado con hechos. Para este propósito, la clase *SMWSemanticData* provee un contenedor básico para manejar conjuntos de hechos que se refieren al mismo sujeto. Esto tiene sentido dado que es el caso más común donde varios hechos se refieren al mismo sujeto, por ejemplo todos los hechos en una página o todos los hechos en una linea de una respuesta de una consulta. Un *SMWSemanticData* contiene un grupo de valores por propiedad, contiene una lista de propiedades y por cada propiedad una lista de valores. Esto refleja el patrón de acceso a las propiedades y evita la duplicación de información. La información contenida en un *SMWSemanticData* puede ser vista como una tripla sujeto-propiedad-valor pero dentro de **SMW** no existe una forma dedicada para la representación de esas triplas.

**Conclusión.**

Una ***Wiki*** es una aplicación web que permite a múltiples usuarios la creación y edición de páginas web de forma colaborativa. Un usuario puede crear un nuevo artículo, eliminar un artículo o editar un artículo dentro de la ***Wiki***. El sistema ***Wiki*** realiza un seguimiento de todas las modificaciones de los artículos y provee herramientas para la organización de la información como las categorías, *templates* o *infobox*.

En la utilización de la Web los usuarios perciben que la información se encuentra inconsistente, desincronizada y desconectada. La ***Semantic Web*** intenta solucionar estos problemas brindando una infraestructura de datos distribuida. La principal idea de la ***Web Semántica*** es soportar una Web distribuida al nivel de datos en lugar de un nivel de representación. Para esto se publica junto con la representación, información estructurada describiendo conceptos reales y sus relaciones. La información estructurada en la ***Web Semántica*** se representa utilizando el lenguaje ***RDF (Resource Description Framework)***. El objetivo de este lenguaje es lograr el intercambio de información en la *Web* manteniendo el significado original de la misma. Luego, con el lenguaje ***OWL (Web Ontology Languaje)*** se intenta obtener un lenguaje para representar información con un balance entre la expresividad y la eficiencia de razonamiento. Para realizar consultas a la información semántica se utiliza el lenguaje de consultas ***SPARQL***. Utilizando este lenguaje se describe un grafo que se compara con las triplas y aquellas que cumplen con las condiciones son retornadas por la consulta.

Una ***Wiki Semántica*** combina las propiedades de las ***Wikis*** tradicionales con las tecnologías de la **Web Semántica.** Se mantiene las propiedades de las ***Wikis*** y se agregan las tecnologías de la **web semántica** como el contenido estructurado, modelos de conocimiento en forma de *ontología*, y búsquedas semánticas. En una ***wiki semántica*** los usuarios agregan al contenido de los *artículos wikis* anotaciones semánticas. Utilizando las anotaciones los usuarios realizan una asociación entre una estructura y el contenido de un *artículo wiki*. La estructura que se genera forma la *ontología* de la ***wiki semántica***.

**Semantic Media Wiki (SMW)** es una de las ***wikis semánticas*** más utilizadas principalmente por su facilidad de uso. La información semántica es ingresada por el usuario en forma de anotaciones semánticas junto al contenido de los artículos. Entre las anotaciones que pueden realizar los usuarios están las categorías son una forma sencilla de anotación que permite a los usuarios clasificar páginas. Las relaciones, que agregan semántica a los *links* existentes, describen como están relacionados dos artículos. Los atributos permiten a los usuarios especificar relaciones entre artículos y objetos que no son artículos.

Las tecnologías descriptas en este capítulo son las involucradas en esta tesis. Es necesario comprender el objetivo de cada una para lograr comprender el objetivo que se tiene con este tesis. Las ***Wikis Semánticas*** pueden ayudar mucho a la organización del contenido generado de forma colaborativa. Sin embargo, dado los problemas que tienen se necesita una herramienta para la edición y mantenimiento de la información semántica. En el siguiente capítulo se describe la estrategia adoptada asistir al usuarios en la evolución semántica.

# Capítulo 3

## Una solución basada en refactorings

*La **invención**, debemos aceptarlo humildemente, no consiste en crear algo de la nada sino a partir del **caos**.*

***Jonathan Lethem.***

## Introducción.

En *Ingeniería de Software* el concepto de *code smell* indica la existencia de un síntoma en el código fuente que puede indicar la presencia de un error mas profundo. En este capítulo se describe el concepto ***Semantic Wiki Bad Smell*** que basado en *code smell* se adapta para ser utilizado en ***Wikis Semánticas***.

Los **refactorings** son utilizados en la *Ingeniería de Software* para mejorar la calidad del software. Se refiere al proceso de realizar cambios persistentes y progresivos a la estructura interna del sistema. Se adapta este concepto para ser utilizado en ***Wikis Semánticas***.

Se presenta el catálogo de ***Semantic Wiki Bad Smells***, cada uno representa un síntoma que debe ser analizado de forma contextual por el usuario. Se describen las situaciones en las cuales realmente ocurre el síntoma y que se debe analizar para determinar si existe un síntoma.

Se presenta el catálogo de ***Semantic Wiki Refactorings***, cada uno representa un proceso de cambios necesarios para eliminar un síntoma en la estructura interna de la ***Wiki Semántica***. Se describen cual es el objetivo de cada ***refactoring*** y cuales son las modificaciones realizadas.

En el presente capítulo, se presenta se solución propuesta. En la *sección 1* se define un ***Semantic Wiki Bad Smell***. En la *sección 2*, se presenta el concepto de ***Semantic Wiki Refactoring***. Luego, en la *sección 3* se presenta el catálogo de ***Semantic Wiki Bad Smells*** y en la *sección 4* se presenta el catálogo de ***Semantic Wiki Refactorings***.

### 1. Estrategia.

La estrategia se basa en dos conceptos claves, los ***Semantic Wiki Bad Smells*** y en ***Semantic Wiki Refactorings***. El primer concepto describe cuales son los síntomas que puede indicar un error mas profundo en la estructura interna de una ***Wiki Semántica***. El segundo concepto describe cuales son los procesos necesarios para eliminar de forma total o parcial los síntomas detectados.

El objetivo es asistir al usuario durante la evolución de la ***Wiki Semántica***. Para lograr este objetivo, se brinda al usuario la información necesaria para la toma de decisiones. Los usuarios deben analizar en forma contextual los síntomas descriptos por los ***Semantic Wiki Bad Smells*** y determinar si es necesario aplicar ***refactorings***. Este proceso no puede ser automatizado debido a que requiere ser analizado en forma contextual.

En las siguientes secciones se describen cuales son los síntomas que los usuarios deben analizar y cuales son los procesos necesarios para eliminarlos. Se proponen los ***refactorings*** que se ejecutan de forma automática para garantizar la no incorporación de nuevos errores en la *ontología.*

### 2. Semantic Wiki Bad Smells.

En *Ingeniería de Software* el concepto *code smell* indica la existencia de un síntoma en el código fuente que puede indicar la presencia de un error mas profundo. Indica una debilidad en el diseño aumentando la posibilidad de introducir errores (*bugs)* o futuras fallas de diseño.

Un ***Semantic Wiki Bad Smell*** es un síntoma en la estructura semántica de una ***Wiki Semántica*** que posiblemente indique un problema mas profundo y sugiere que un ***refactoring*** debe ser aplicado. La aplicación de un ***refactoring*** resulta tan importante como determinar *cuando* es necesario aplicar ***refactorings*** y *cuando* dejar de hacerlo. La presencia de un ***Semantic Wiki Bad Smell*** no necesariamente indique un error, debe ser analizado en contexto para determinar si es necesario aplicar un ***refactoring***.

Para asistir al usuario en la aplicación de ***refactorings***, cada ***bad smell*** cuenta con un conjunto de ***refactorings*** que deben ser tenidos en cuenta para eliminar el síntoma. Debido a que la decisión de aplicar ***refactorings*** es contextual, se debe analizar con información del contexto para determinar si el error existe. Por ejemplo, se debe tener en cuenta la cantidad de artículos promedios que tienen las categorías para determinar *cuando* una categoría es demasiado grande.

Las ***wikis*** son aplicaciones que evolucionan con rapidez, por lo cual el objetivo es asistir de forma continua al usuario para mejorar la calidad de la *ontología.* La asistencia consiste en contar con una herramienta para detectar posibles errores. El mecanismo de detección puede involucrar la ejecución de una consulta semántica a la base de datos de conocimiento, o puede ser necesario métodos mas complejos como técnicas de *data minning*. Siempre que sea posible se brinda un mecanismo automático para detectar ***Bad Smells***.

Supongamos que tenemos la categoría 'Ciudad' y la categoría 'Metrópolis'. Al ver cuales son los

*artículos wikis* con los que cuenta cada categoría, vemos que todos los artículos que pertenecen a 'Metrópolis' también pertenecen a 'Ciudad'. Existe un ***Bad Smell*** que indica que esto puede ser un síntoma de un error. El ***bad smell*** '*Twins Categories*' indica que si estos sucede, puede ser un error por parte de los usuarios.

El ***Bad Smells*** *Twin categories Bad Smell* describe el caso donde dos categorías aparecen en el mismo recurso de forma muy frecuente. Una de las formas de detectar el ***bad smell***, es si dos categorías tienen la misma semántica pero diferentes nombres. Este caso es intrínseco a la forma colaborativa de las ***wikis semánticas***. Se puede llegar a esta situación a causa de que un usuario no conoce una categoría existente, o puede ser porque pertenecen a diferentes materiales profesionales o hablan diferentes idiomas. La consecuencia es que la categoría es duplicada.

El segundo caso donde puede aparecer este ***bad smell*** es cuando la categoría gemela define una relación de subcategoría. Por la falta de la relación de subcategoría entre dos categorías sucede que el mismo recurso es categorizado con dos categorías diferentes.

### 3. Semantic Wiki Refactorings.

En *Ingeniería de Software* los ***refactorings*** se utilizan para mejorar la calidad del software. El término ***refactoring*** se refiere al proceso de realizar cambios persistentes y progresivos a la estructura interna del sistema. Los cambios producidos por los ***refactorings*** no modifican el comportamiento externo, pero mejoran la calidad del diseño [3]. La ventaja radica en la forma automática de ejecución, que garantiza la no incorporación de nuevos errores al sistema.

El concepto de ***refactorings*** de *Ingeniería de Software* se adapta para su utilización en ***Wikis Semánticas***. Un ***Semantic Wiki Refactoring*** especifica transformaciones en la estructura básica de conocimiento de una ***Wiki Semántica***. Esto permite mejorar la calidad implícita de la *ontología*. Los ***refactorings*** describen una forma ordenada de realizar transformaciones apropiadas para evitar nuevas inconsistencias, redundancias y otros problemas de calidad.

Siguiendo con el ejemplo del *capítulo 1*, detectamos que es necesario realizar la relación de subcategoría entre las categorías '*Ciudad*' y '*Ciudad Capital*'. Es claro, que '*Ciudad Capital*' debe ser subcategoría de '*Ciudad*'. Para realizar esta relación en la forma apropiada se deben realizar dos tareas. La primera, es hacer explicita la relación de subcategoría entre las dos categorías. Luego, de las páginas que pertenecen a las dos categorías se debe eliminar la categorización de la supercategoría. Esto es debido a que la relación esta dada de forma implícita (por la relación de subcategoría). Realizar esta tarea sin la utilización de un ***semantic refactoring*** implica la modificación en primer lugar del artículo de la subcategoría, luego se debe modificar todas aquellos artículos que pertenecen a ambas categorías y eliminar la categorización de la supercategoría.

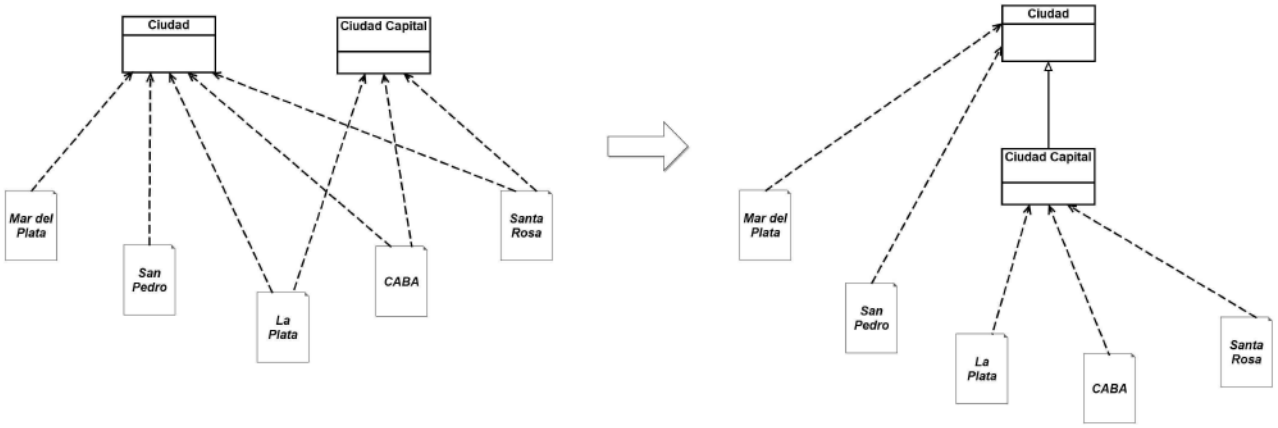


Imagen 14. Aplicación de un ***refactoring***.

En la imagen vemos el estado antes y después de la aplicación del ***refactoring***. En la imagen de la izquierda vemos que las dos categorías no tienen ninguna relación. También podemos ver que aquellas páginas que son '*Ciudad*' y también son *'Ciudad Capital'* pertenecen a ambas categorías de forma directa. Es decir, para pertenecer a la categoría los usuarios deben agregar la anotación que indica la categoría. En la imagen de la derecha vemos el estado luego de aplicar el ***refactoring***. Las categorías han sido relacionadas por medio de la relación subcategoría. Esto establece que si una página pertenece a la subcategoría, también pertenece a la supercategoría. Por este motivo vemos la simplificación en la categorización de las páginas. Vemos que las páginas que pertenecen a ambas categorías solo deben pertenecer a la subcategoría.

La aplicación de un ***refactoring*** puede realizarse de forma *manual* aunque es mas probable la incorporación de nuevos errores a la solución del problema original. Por lo cual, contar con una herramienta automática para la modificación de todos los artículos es primordial. Ahora supongamos la incorporación de una nueva categoría llamada '*Espacio geografico*', que tendría que ser supercategoría de las dos categorías anteriores. Nuevamente se tendría que *refactorizar* para lograr las modificaciones necesarias. La evolución es una tarea constante y se le debe dar al usuario las herramientas necesarias para mejorar la calidad de la *ontología* en todas las etapas de su evolución.

## 4. Catálogo de Semantic Wiki Bad Smells.

Se presenta el catálogo de ***Semantic Wiki Bad Smells***, *c*ada uno describe un síntoma en la estructura interna de la ***Wiki Semántica*** y ayuda a asistir al usuario en la toma de decisiones para eliminar el síntoma detectado. El usuario debe analizar si es necesario y cuál ***refactoring*** se debe aplicar.

Se define un ***Bad Smells*** con un nombre, una descripción informal del síntoma y un conjunto de ***Semantic Wiki Refactorings*** que se proponen para eliminar de forma total o parcial el ***bad smell***. Finalmente se presenta un ejemplo con los escenarios donde puede aparecer un ***bad smell***.

En los casos que es posible se define un mecanismo de detección automático. Se define formalmente el

mecanismo de detección mediante una consulta **SPARQL**. Luego, este mecanismo puede ser adaptado a las tecnologías que diferentes ***wikis semánticas*** proveen.

Para realizar la definición formal se requiere de un modelo sobre el cuál realizar las consultas por lo cuál se propone utilizar el siguiente modelo [4]:

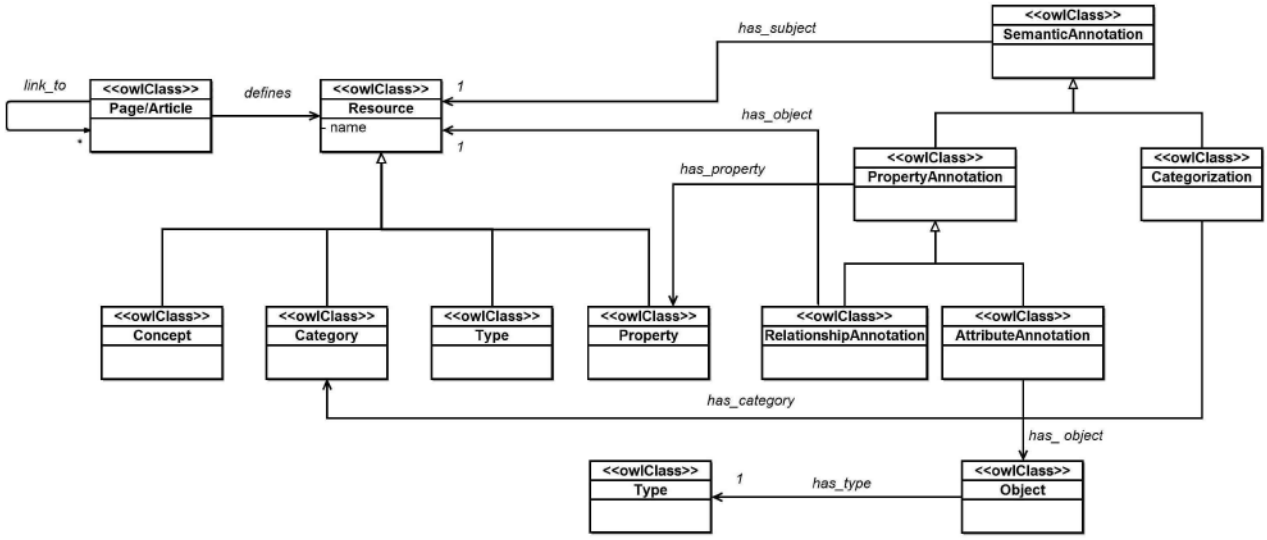


Imagen 15. Modelo.

Un *Recurso (Resource)* es todo aquello que puede ser descripto en una wiki: un *Concepto (Concept)*, que representa individuos en la *wiki*; *Categorías (Category)*, que permite agrupar recursos; *Propiedad (Property)*, que describe relaciones semánticas entre recursos; y *Tipos (Type)*, que son utilizados para especificar el tipo de objeto de las propiedades. Cada uno de estos recursos puede tener una *Página* o *Artículo Wiki* que lo describe. Las *Páginas* también pueden estar relacionadas con otras páginas utilizando *links* sin tipo.

Las Anotaciones Semánticas (*Semantic Annotation*) están subclasificadas en *Categorización (Categorization)* y *Anotación de Propiedad (Property Annotation)*. Una *Categorización* describe una relación de pertenencia de un recurso a una categoría. Una relación de subcategoría entre categorías es definida cuando el recurso del sujeto (*subject*) de una categorización es una categoría. En este caso, la categoria sujeto es subcategoría del objeto categoría.

Una *Anotación de Propiedad (Property Annotation)* es compuesta por tres elementos: un *sujeto* (*subject*), que puede ser cualquier recurso; una *propiedad*, que es un individuo de *Propiedad*; y un *objeto*. Las *Anotaciones de Relaciones* son aquellas propiedades donde el objeto es otro recurso, mientras que el objeto *Anotación de Atributo (Attribute Annotations)* son valores finales de un tipo de datos.

**Catálogo de Semantic Wiki Bad Smells:**

| | Nombre Bad Smell | Descripción |
|---|---|---|
| **Category Bad Smells.** | | |
| 1 | **Concept too categorized** | Un *artículo wiki* pertenece a demasiadas categorías. |
| 2 | **Large category** | Una categoría contiene demasiados *artículos wiki*. |
| 3 | **Twin Categories.** | Dos categorías son utilizadas muy seguido en un mismo *artículo wiki*. |
| 4 | **Super Category with more articles than his subcategory.** | Una categoría que es supercategoría de otra tiene mas *artículos wiki* que una subcategoría. |
| **Property Bad Smells.** | | |
| 5 | **Concept too annotated** | Un *artículo wiki* contiene demasiadas anotaciones. |
| 6 | **Large Property.** | Una propiedad es utilizada demasiadas veces. Los usuarios utilizan la propiedad en muchos *artículos wiki*. |
| 7 | **Twin properties.** | Dos propiedades aparecen juntas en un mismo *artículo wiki* de forma muy frecuente. |
| 8 | **Divergent property.** | La propiedad es utilizada con diferentes valores que divergen en rango y dominio. |
| 9 | **Resource with no semantic annotations.** | Un *artículo wiki* no contiene anotaciones semánticas. |

Tabla 7. Catálogo de ***Semantic Wiki Bad Smells***.

**1. Concept too categorized.**

**Descripción:**

Un *artículo wiki* pertenece a demasiadas categorías de forma directa, no se tienen en cuenta las categorías a las que pertenece el artículo por herencia. Por alguna razón los usuarios agregan el artículo a muchas categorías. Que esto suceda no indica necesariamente un error, sino que se debe analizar el motivo por el cuál sucede. Para determinar cuantas categorías son *demasiadas* se debe hacer un análisis comparativo con el resto de los artículos o considerando la cantidad de categorías que existen en la ***wiki***.

El artículo puede pertenecer a categorías que deberían estar relacionadas como subcategorías. Si las categorías no están relacionadas como subcategorías el artículo debe pertenecer de forma directa a todas ellas. Si las categorías están organizadas en una jerarquía de categorías con pertenecer a una subcategoría el artículo hereda la categorización de las supercategorías de forma automática.

Las categorías pueden estar mal utilizadas por los usuarios. Es necesario comprende el criterio de

categorización de cada una de ellas. Posiblemente se detecte una categoría mas específica a la cual el artículo debe pertenecer.

Un artículo puede describir mas de un concepto real. Al describir mas de un concepto califica para pertenecer a más categorías. Siendo un artículo muy extenso puede estar definiendo mas de un concepto real y por lo cuál pertenecer a muchas categorías.

**Mecanismo de detección**:

```
SELECT ?resource        count(?categorization)
WHERE {
                ?categorization  has_subject            ?resource
                ?categorization  has_category    ?category
        }
GROUP BY        ?resource
HAVING          (count(?categorization) > PARAM)
LIMIT   1
```

**Refactorings Asociados**:

1. ***Create subcategory refactoring.***

   Puede suceder que las categorías a la que pertenece el artículo no están relacionadas a través de una subcategorización. Utilizando el ***refactoring*** propuesto se puede relacionar las categorías del artículo y luego eliminar la categorizaciones de las supercategorías utilizando ***Delete category relationship refactoring.***

2. ***Change category refactoring.***

   Si el artículo pertenece a muchas categorías puede ser producto que no esta bien categorizado. Se puede mover el artículo hacia una categoría que es mas específica y no es necesario que pertenezca a muchas categorías.

3. ***Delete category relationship refactoring.***

   Si el artículo pertenece a muchas categorías pude suceder que no se haya comprendido el propósito de alguna de las categorías y se deban eliminar las categorizaciones.

**Ejemplo:**

Supongamos que se cuenta con las categorías 'Ciudad', 'Capital Provincial' , 'Ciudad Universitaria' y 'Cabecera Partido' y que no existe ninguna relación de subcategoría entre ellas. Se cuenta con un artículo para la ciudad de 'La Plata'. Este artículo pertenece a las categorías 'Ciudad', 'Capital Provincial', 'Ciudad Universitaria', 'Cabecera Partido'.

Como las categorías no cuentan con una relación de subcategoría entre ellas, se debe agregar la categorización para cada una de las categorías. Si las categorías estarían relacionadas el artículo pertenece a

las supercategorías de forma automática y la cantidad de categorías a las que pertenece el artículo de forma directa sería menor.

Aplicando en forma sucesiva ***Create subcategory refactoring*** se genera un árbol de categorías, luego utilizando ***Delete category relationship refactoring*** se elimina del artículo las categorizaciones de las supercategorías.

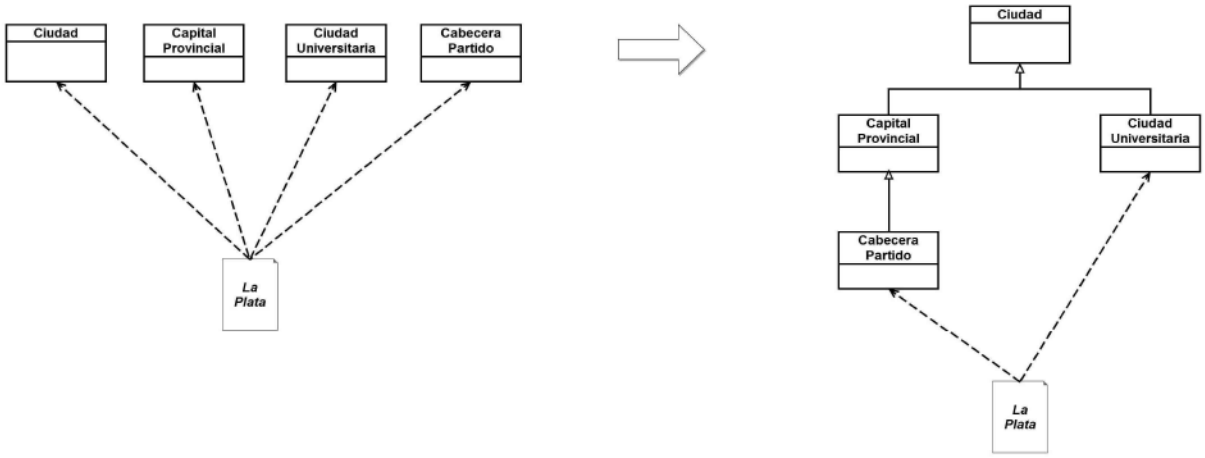


Imagen 16. Categorías del artículo de la ciudad de 'La Plata'.

En la imagen anterior vemos como el artículo de la ciudad de 'La Plata' pasa de tener cuatro categorizaciones a solo dos. Por la relaciones de subcategoría de las categorías a las que pertenece de forma directa sigue siendo parte de las mismas categorías.

### 2. Large category

**Descripción:**

Una categoría contiene demasiados *artículos wiki*. Se espera que todas las categorías de una ***wiki*** contengan una cantidad equilibrada de artículos. Si una categoría contiene demasiados artículos en comparación con el resto de las categorías se debe analizar el motivo. Puede suceder que sea una anomalía provocada por el dominio de la ***wiki*** y no requiera ninguna modificación.

Para determinar la cantidad de artículos que pertenecen a una categoría se tiene en cuenta solamente los artículos que forman parte de la categoría de forma directa. Es decir, no se tiene en cuenta los artículos que pertenecen a la categoría por pertenecer a una subcategoría.

La categoría puede contener muchos artículos debido a que está representando más de un concepto real de categorización. De esta manera agrupa artículos con mas de una característica en común, es decir no cumple con el objetivo de una categoría.

La categoría puede ser mal interpretada. Los usuarios no comprenden el criterio de agrupación que define la categoría por lo cual agregan artículos que no deberían pertenecer. Siempre se debe contar con un artículo detallado con los criterios que definen que un *artículo wiki* pertenezca a la categoría. Puede suceder que

en un momento de la evolución de la ***wiki*** no existan otras categorías con criterios similares, luego de la creación de estas categorías debe actualizarse las categorías a las que pertenecen los artículos.

Al analizar la situación podemos detectar que se está representando una categoría muy abstracta y se necesite realizar una subcategorización.

Para determinar cuando una categoría tiene demasiados *artículos wiki* hay que conocer la ***wiki***, por lo cual el proceso no se puede automatizar y el usuario debe analizar cuando el ***bad smell*** indica un error.

**Mecanismo de detección:**

```
SELECT ?category        count(?category)
WHERE {
                ?categorization  has_category    ?category
                ?categorization  has_subject     ?subject
                }
GROUP BY ?category
HAVING          (count(?category) > PARAM )
LIMIT   1
```

**Refactotings Asociados:**

1. **Split Category Refactoring.**

   La categoría puede contener muchos artículos debido a que está representando más de un concepto real de categorización. Se debe dividir la categoría creando una nueva categoría y agregando a cada categoría un subconjunto de los artículos de la categoría original.

2. **Change Category Refactoring.**

   Puede contener muchos artículos debido a que los usuarios no conocen otras categorías a la cual deberían pertenecer los artículos. Por lo cual, se debería mover los artículos hacia otras categorías existentes.

3. ***Create subcategory refactoring.***

   La categoría puede ser muy abstracta y requiere hacer una subcategorización. Mediante este ***refactoring*** se establece la relación de subcategoría.

**Ejemplo:**

Supongamos que no existe ninguna relación de subcategoría entre las categorías 'Ciudad','Capital Provincial', 'Ciudad Universitaria', 'Cabecera Partido'.

Luego, todas los artículos que son 'Capital Provincial', 'Ciudad Universitaria' y 'Cabecera Partido' también pertenecen a la categoría 'Ciudad' de forma directa. Por lo cual esta última categoría contendrá demasiados artículos. Si existiera la relación de subcategorías antes mencionadas, la categoría 'Ciudad'

solamente contendría artículos que no son 'Capital Provincial', 'Ciudad Universitaria' y 'Cabecera Partido'. Se detecta el síntoma que una categoría contiene demasiados artículos y luego se detecta el error que las categorías no están relacionadas como subcategorías.

Otro motivo podría ser que la categoría esta representando mas de un criterio de categorización y se requiera dividir las categorías. Por ejemplo, si tenemos una categoría que representa 'Edificio', y esta categoría contiene muchos artículos es necesario dividir la categoría. Se puede sugerir la división de la categoría con una categoría llamada 'Edificio Publico'. En la siguiente imagen se ven los cambios necesarios:

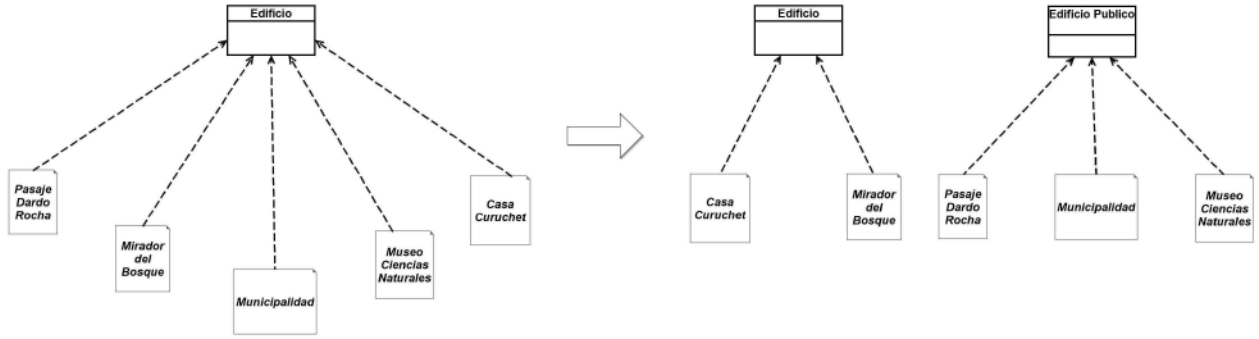


Imagen 17. ***Split Category Refactoring***.

En primer lugar vemos como la categoría 'Edificio' cuenta con cinco artículos wiki, luego al dividir la categoría cuenta con dos artículos, y la nueva categoría cuenta con tres artículos. De esta forma la cantidad de artículos de cada categoría se encuentra mas equilibrada. El usuario finalmente debe analizar si se requiere realizar una relación de subcategorías entre las dos categorías involucradas.

**3. Twin Categories.**

**Descripción:** Dos categorías son utilizadas muy seguido en un mismo *artículo wiki*. Existen demasiados artículos que pertenecen a las dos categorías.

Esta forma de utilizar las categorías puede indicar que las dos categorías representan el mismo concepto. Pueden diferir en el nombre pero el concepto de agrupación es el mismo. Por lo cual representan una misma categoría con diferentes nombres.

**Mecanismo de detección:**

```
SELECT DISTINCT ?categoriaA ?categoriaB
WHERE {     ?categorizacionA    has_subject     ?subjectA
            ?categorizacionB    has_subject     ?subjectA
            ?categorizationA    has_category    ?categoriaA
            ?categorizationB    has_category    ?categoriaB.

FILTER ((?categorizacionA != ?categorizacionB)
            && (?categoriaA != ?categoriaB))
            }

GROUP BY ?categoryA  ?categoryB
HAVING      (count(*) > PARAM )
```

**Refactorings Asociados:**

1. **Join Category Refactoring.**

   Dadas dos categorías se fusionan formando una sola. Los artículos pertenecientes a las dos categorías forman parte de la categoría fusionada.

2. ***Create Subcategory Relationship Refactoring.***

   Una categoría puede ser subcategoría de la otra y esta relación no esta especificada. Realizando la relación se subcategorías se elimina una de las categorizaciones en aquellos artículos donde están presente las dos categorizaciones.

**Ejemplo:**

Supongamos que tenemos dos categorías, una llamada 'Ciudad' y otra llamada 'Urbe', mediante el ***bad smell twin categories*** detectamos que existen demasiados artículos que forman parte de la categoría 'Ciudad' y también de la categoría 'Urbe'. Luego de leer los artículos que describen a cada una de las categorías se determina que representan la misma categoría.

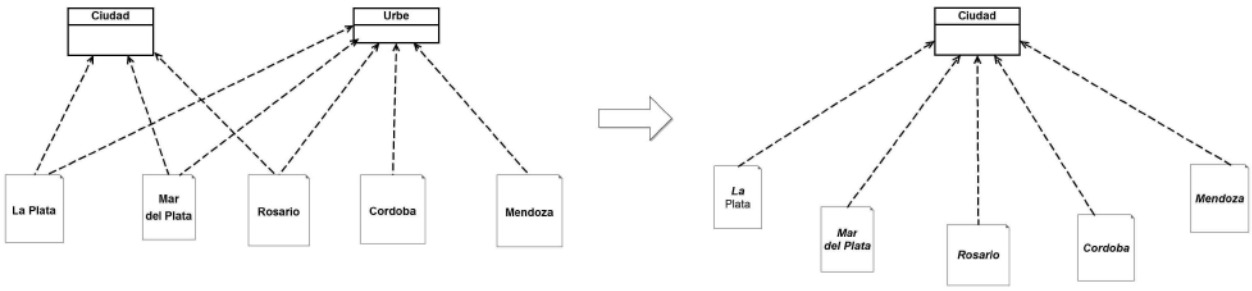


Imagen 18. ***Join Category Refactoring.***

En la imagen vemos como los artículos pertenecen a dos categorías que en realidad representan lo mismo. Al detectar este síntoma se unifican las categorías, luego los artículos pertenecen a una sola categoría. La decisión de unir dos categorías la toma el usuario que utiliza la herramienta. Puede suceder que para el mismo caso se requiera crear una relación de subcategoría, como se sugiere en los ***refactorings*** a tener en cuenta para eliminar el ***Bad Smell***.

**4. Super Category with more articles than his subcategory.**

**Descripción:**

Una categoría que es supercategoría de otra y contiene más *artículos wiki* que una de sus subcategoría. Es una buena práctica que las subcategorías sean las que contienen los artículos. Puede significar que es necesario modificar la estructura de la jerarquía de categorías. Puede suceder que una supercategoría no sea tan abstracta como se pensaba o que una subcategoría no sea tan específica. Si los usuarios no conocen las subcategorías podrían incluir los artículos en las categorías mas abstractas. Si una supercategoría contiene mas

artículos que alguna de sus subcategorías puede suceder que sea necesario crear una nueva subcategoría.

**Refactorings Asociados:**

1. **Hierarchy pull Up category Refactoring.**

   La jerarquía de categorías puede estar mal diseñada. Se modifica la estructura de categorías subiendo la categoría un nivel en el árbol de jerarquías.

2. **Hierarchy pull down category Refactoring.**

   La jerarquía de categorías puede estar mal diseñada. Se modifica la estructura de categorías bajando la categoría un nivel en el árbol de jerarquías.

3. ***Change category refactoring.***

   Las supercategorías deberían ser mas abstractas que las subcategorías. Se logra equilibrar la cantidad de artículos simplemente moviendo los artículos hacia una categoría mas específica que por consiguiente será una subcategoría.

4. ***Create subcategory refactoring.***

   Puede suceder que se requiera la incorporación de una nueva categoría a la jerarquía de categorías.

**Ejemplo:**

Supongamos que tenemos la categoría 'Ciudad' y la subcategoría 'Ciudad Capital'. Si la categoría no tiene otras subcategorías es claro que la supercategoría va a tener mas artículos que la subcategoría. De esta forma el ***bad smell*** detecta un síntoma que indica la posible presencia de un error. Para solucionar este síntoma detectado se pueden crear nuevas categorías o incluso realizar la relación de subcategoría con otras categorías que existen pero no son subcategorías de 'Ciudad'.

**5. Concept too annotated.**

**Descripción:**

Un *artículo wiki* contiene demasiadas anotaciones. En comparación con el resto de los artículos los usuarios realizaron demasiadas anotaciones.

El artículo contiene propiedades que deberían estar relacionadas como subpropiedades. Si las propiedades no están relacionadas como subpropiedades los usuarios también deben agregar las anotaciones de las superpropiedades.

Dentro del artículo detectamos que existen propiedades que representan el mismo concepto real. Dado que existen propiedades duplicadas que difieren solamente en el nombre de la propiedad los usuarios deben agregar mas anotaciones a los artículos.

Un artículo puede estar representando más de un concepto real. Dentro del artículo se describen demasiados conceptos reales lo que genera que se utilicen demasiadas anotaciones.

**Refactorings Asociados:**

1. **Add subproperty refactoring.**

   Dentro del artículo se encuentran propiedades que tendrían que ser subpropiedades entre ellas. Si se genera la relación de subpropiedad entre las propiedades que se utilizan, no es necesario agregar las propiedades de las superpropiedades.

2. **Join property refactoring.**

   Dentro del artículo encontramos propiedades que tienen el mismo nombre pero representan la misma relación. Las propiedades se deben unificar y así el artículo tendrá menos anotaciones.

**Ejemplo:**

Supongamos que tenemos un artículo que representa a la ciudad de 'La Plata'. Luego ve que tiene las propiedades 'población', 'habitantes', 'provincia', 'estado', 'altitud', 'metros sobre el nivel del mar'.

Luego de analizar los artículos de las propiedades llegamos a la conclusión que debemos unificar las propiedades utilizando **Join property refactoring**.

Se unifica 'población' con 'habitantes' dado que ambas representan la cantidad de habitantes de la ciudad. Se unifica 'provincia' con 'estado' dado que la propiedad estado esta utilizada de forma incorrecta ya que ambas intentan indicar a la provincia a la que pertenece la ciudad. Finalmente se unifica 'altitud' y 'metros sobre el nivel del mar' dado que ambas representan a que altitud se encuentra la ciudad.

**6. Large Property.**

**Descripción:**

Una propiedad es utilizada demasiadas veces. Los usuarios utilizan la propiedad en demasiados *artículos wiki*. Comparando la cantidad de veces que los usuarios utilizan el resto de las propiedades podemos determinar cuando una propiedad es utilizada demasiadas veces.

Si los usuarios utilizan una propiedad demasiadas veces puede indicar que la propiedad represente mas de un concepto real. Se debe especificar con claridad en el artículo de la propiedad cual es el objetivo de la misma.

La propiedad puede estar representando una propiedad muy abstracta y por consiguiente es utilizada en demasiados artículos.

**Mecanismo de detección:**

```
SELECT DISTINCT ?propertyA     count(?propertyA)
WHERE {         ?propertyAnnotation     has_property     ?propertyA
                }
GROUP BY ?propertyA
HAVING          (count(?propertyA) > PARAM )
```

**Refactorings asociados:**

1. **Split Property.**

   Se divide a la propiedad en dos propiedades. Se crea una nueva propiedad por consiguiente la cantidad de artículos que utilizan la propiedad original se reduce.

2. **Add subproperty refactoring.**

   La propiedad esta representando una propiedad abstracta y requiere una subclasificación de la misma. Luego, la cantidad de artículos que la utilizan de forma directa se reduce.

**Ejemplo:**

Supongamos que tenemos una propiedad llamada 'Pareja de' y detectamos que es utilizada en demasiados artículos. Luego, nos damos cuenta que es necesario dividir la propiedad debido a que se está utilizando la propiedad para representar las personas que están casadas. Luego, se divide la propiedad creando una nueva llamada 'Casado con'. De esta forma la cantidad de artículos que utilizan la propiedad se reduce.

**7. Twin properties.**

**Descripción:**

Dos propiedades aparecen juntas en un mismo *artículo wiki* de forma muy frecuente. Dadas dos propiedades de la ***wiki***, si sucede de forma frecuente que cuando los usuarios utilizan una propiedad también utilizan la segunda propiedad puede suceder que una propiedad es subpropiedad de la segunda. Dos propiedades pueden aparecer juntas debido a que tienen una relación de subpropiedad entre ellas. Si esto sucede la relación no esta establecida se debe establecer para lograr mayor integridad en los datos.

Las dos propiedades pueden estar representando el mismo concepto real pero con diferentes nombres. Si bien se detecta que son utilizadas de forma conjunta puede suceder que diferentes usuarios utilicen diferentes propiedades cuando en realidad son la misma propiedad.

**Mecanismo de detección**:

```
SELECT DISTINCT ?propertyA ?propertyB

WHERE {
        ?propertyAnnotationA has_subject      ?subjectA
        ?propertyAnnotationB has_subject      ?subjectA
        ?propertyAnnotationA   property       ?propertyA
        ?propertyAnnotationB   property       ?propertyB.

FILTER ((? propertyAnnotationA != ?propertyAnnotationB)
        && (?propertyA != ?propertyB))
        }

GROUP BY ? propertyA ?propertyB
HAVING      (count(*) > PARAM )
```

**Refactorings Asociados**:

1. **Add Subproperty Refactoring.**

   Si dos propiedades aparecen juntas muchas veces puede ser indicio de que las propiedades tienen una relación de subpropiedades entre ellas.

2. **Join Property Refactoring.**

   Si dos propiedades aparecen juntas muchas veces puede ser indicio de que las propiedades en realidad describen el mismo concepto.

**Ejemplo:**

Supongamos que contamos con una propiedad llamada 'casado con' y con otra propiedad llamada 'en matrimonio con'. El ***bad smell*** detecta que las propiedades son utilizada con frecuencia en un mismo artículo lo que indica un posible error. Luego de analizar los artículos de cada una de las propiedades llegamos a la conclusión de que representan la misma propiedad. Se utiliza **Join Property Refactoring** para unificar las dos propiedades.

**8. Divergent property.**

**Descripción:**

La propiedad es utilizada con diferentes valores que divergen en rango y dominio. Los valores con los cuales los usuarios utilizan la propiedad son diferentes entre sí. Se debe analizar si el artículo de la propiedad esta establecido cuál es el objetivo de la propiedad y cuales son los valores posibles. Puede suceder que los usuarios no comprendan como deben utilizar la propiedad o cual es su objetivo.

La propiedad puede estar representando mas de un concepto real. Si los usuarios utilizan la propiedad con diferentes valores, sucede que la propiedad está siendo utilizada para representar diferentes propiedades.

**Refactorings Asociados:**

1. **Split property refactoring.**

   La propiedad esta representando mas de un concepto. Se debe crear una o varias nuevas propiedades para la correcta representación de la información.

**Ejemplo:**

Supongamos que contamos con una propiedad llamada 'clima'. La propiedad es utilizada en artículos que describen ciudades. Para esta misma propiedad los usuarios utilizan los siguientes valores: 'templado', 'lluvioso', 'soleado', '21 grados', '10 grados'. Es claro que los valores de la propiedad divergen en rango y dominio. Se ve la necesidad de crear una nueva propiedad que represente los diferentes tipos de valores. Analizando los artículos se llega a la conclusión de dividir la propiedad y crear una nueva llamada 'temperatura en grados' y continuar con la propiedad 'clima' para indicar que clima predomina en la zona o ciudad.

**9. Resource with no semantic annotations.**

**Descripción:**

Un *artículo wiki* no contiene anotaciones semánticas. Si un artículo no contiene información semántica no es posible utilizar todos los beneficios de una ***Wiki Semántica***.

La no incorporación de información semántica puede indicar que es necesario la generación de nuevas anotaciones. También puede suceder que no exista una categoría para el *artículo wiki* o que sea necesario crear nuevas propiedades.

**Mecanismo de detección**:

```
SELECT DISTINCT ?resourceA
WHERE {     ?semanticAnnotation     has_subject     ?resourceA
            }
GROUP BY    ?resourceA
HAVING      (count(?resourceA) == 0)
```

**Refactorings Asociados**:

1. **Add category refactoring.**

   Como mínimo una página debe contar con la categoría a la que pertenece. Con este ***refactoring*** se agrega la categoría a la cual debería pertenecer la pagina.

**Ejemplo:**

Supongamos que contamos con un artículo que describe la ciudad de La Plata. Si este artículo no

contiene información semántica es claramente un indicio de algún error o la necesidad de la generación de nueva información semántica.

### 5. Catálogo de Semantic Wiki Refactorings.

Se presenta el catálogo de ***Semantic Wiki Refactorings***. Cada uno se presenta con un nombre, una descripción de cual es su comportamiento, los parámetros que necesita y una implementación (en pseudocódigo) de cada uno de ellos. Cada ***refactoring*** cuenta con un nombre representativo de su comportamiento. Dada la implementación propuesta en esta tesis, se utilizan los nombres con el dialecto propio de ***Semantic Media Wiki (SMW)***. Es claro, que todo el dialecto puede ser generalizado para otras ***wikis***.

El catálogo se divide en dos secciones básicas: *Category Refactorings* y *Properties Refactorings*. Se propone la división basada en los dos conceptos más importantes en la organización de información dentro de una ***Wiki Semántica***.

| | Nombre Refactoring | Descripción |
|---|---|---|
| | ***Category Refactorings*** | |
| **1** | **Add category relationship refactoring.** | Dado un *artículo wiki* se agrega el artículo a la categoría indicada. |
| **2** | **Delete category relationship refactoring.** | Dado un *artículo wiki* se elimina la categorización de la categoría indicada. |
| **3** | **Create subcategory relationship refactoring.** | Agrega una relación de subcategoría entre dos categorías existentes. |
| **4** | **Delete subcategory relationship refactoring.** | Elimina una relación de subcategoría existente entre dos categorías. |
| **5** | **Change category refactoring.** | Dado un *artículo wiki* con una categorización, se elimina una categorización y se agrega una nueva. |
| **6** | **Hierarchy pull up category refactoring.** | Dada una categoría, sube un nivel en el árbol jerárquico de las categorías. |
| **7** | **Hierarchy pull down category refactoring.** | Dada una categoría, baja un nivel en la estructura jerárquica de las categorías. |
| **8** | **Join category refactoring.** | Dadas dos categorías se fusionan formando una sola. Todos los artículos de la segunda categoría pasan a la primer categoría. |

| | | |
|---|---|---|
| **9** | **Split category refactoring.** | Dada una categoría se divide en dos categorías. Se crea una nueva categoría y un subconjunto de artículos de la categoría pasan a la nueva categoría. |
| | ***Properties Refactorings*** | |
| **10** | **Add subproperty refactoring.** | Dadas dos propiedades crea una relación de subpropiedad entre ellas. |
| **11** | **Delete subproperty refactoring.** | Dadas dos propiedades, una subpropiedad de la otra, se elimina la relación de subpropiedad entre ellas. |
| **12** | **Pull up property refactoring.** | En una jerarquía de propiedades, la propiedad seleccionada sube un nivel en árbol de jerarquías. |
| **13** | **Pull down property refactoring.** | En una jerarquía de propiedades, la propiedad seleccionada se baja un nivel en el árbol de jerarquías. |
| **14** | **Join property refactoring** | Dadas dos propiedades se unen para formar una única propiedad. |
| **15** | **Split property refactoring** | Dada una propiedad existente se divide para formar dos propiedades. Se crea una nueva propiedad. |

Tabla 8. Catálogo de ***Refactorings***.

Category Refactorings:

**1. Add category relationship refactoring.**

**Descripción:**

Dado un *artículo wiki* se agrega el artículo a la categoría indicada. El objetivo de este ***refactoring*** es agregar una categorización al artículo. La categoría contiene un nuevo artículo.

En el estado inicial existe un artículo y una categoría, donde el artículo puede pertenecer a otras categorías, pero no pertenece a la categoría indicada. En el estado final el artículo pertenece a la categoría indicada.

**Parámetros:**

- Título o nombre del artículo al cual se quiere agregar la categorización.
- Nombre de la categoría a la cual se debe agregar.

**Implementación:**

1. Seleccionar el artículo indicado.

2. Agregar la categorización al artículo indicado.

**Ejemplo:**

Supongamos que tenemos un artículo que representa a la ciudad de La Plata. Luego decidimos crear una categoría que agrupa a las ciudades que cuentan con universidades llamada 'Ciudad Universitaria'. Mediante este ***refactoring*** podemos agregar al artículo de la ciudad de La Plata a la categoría 'Ciudad Universitaria'.

**2. Delete category relationship refactoring.**

**Descripción:**

Dado un artículo se elimina la categorización de la categoría indicada. El objetivo de este ***refactoring*** es eliminar una categorización del artículo.

En el estado inicial existe un artículo que pertenece a una categoría indicada. El artículo puede pertenecer a otras categorías. En el estado final el artículo no pertenece a la categoría indicada.

**Parámetros:**

- Título de la pagina a la cual se le quiere eliminar la categorización.
- Nombre de la categoría que se quiere eliminar.

**Implementación:**

1. Seleccionar el artículo indicado.
2. Eliminar la categorización de la página de la categoría que es subcategoría.

**Ejemplo:**

Supongamos que tenemos un artículo que describe la ciudad de La Plata y que pertenece de forma directa a las categorías 'Ciudad' y 'Ciudad Capital' las cuales son una subcategoría de la otra. Debido a la relación de subcategoría decidimos eliminar la categorización de forma directa de la supercategoría. Mediante este ***refactoring*** se procede a eliminar la categoría 'Ciudad' del artículo de La Plata.

**3. Create subcategory relationship refactoring.**

**Descripción**:

Agrega una relación de subcategoría entre dos categorías existentes. El objetivo de este ***refactoring*** es establecer una relación de subcategoría entre las dos categorías.

En el estado inicial existen dos categorías sin relación entre ellas. En el estado final la primer categoría es subcategoría de la segunda categoría. Los artículos que pertenecen a ambas categorías se elimina la

categorización de la supercategoría. Si la supercategoría de la nueva supercategoría también es supercategoría de la nueva subcategoría se elimina esa relación de subcategoría.

**Parámetros**:

- Nombre de la categoría que se define como subcategoría
- Nombre de la categoría que se define como supercategoría.

**Implementación**:

1. Agregar en la página de la subcategoría la categorización sobre la supercategoría.
2. De las páginas que pertenecen a ambas categorías, eliminar la categorización de la supercategoría.
3. Para cada una de las supercategorías de la nueva subcategoría, si también es supercategoría de la nueva supercategoría, eliminar la relación de subcategoría con la nueva subcategoría.

**Ejemplo:**

Supongamos que tenemos la categoría 'Ciudad' con las subcategorías 'Cabecera Partido', 'Capital Provincial' y 'Ciudad Universitaria'. Luego, nos damos cuenta que siempre que una ciudad es 'Capital Provincial' obviamente también es 'Cabecera Partido'. Por lo cual, decidimos hacer la primera subcategoría de la segunda. En este caso como la supercategoría de 'Capital Provincial' es 'Ciudad' que también es supercategoría de 'Cabecera Partido' debemos eliminar la relación de subcategoría entre 'Capital Provincial' y 'Ciudad'.

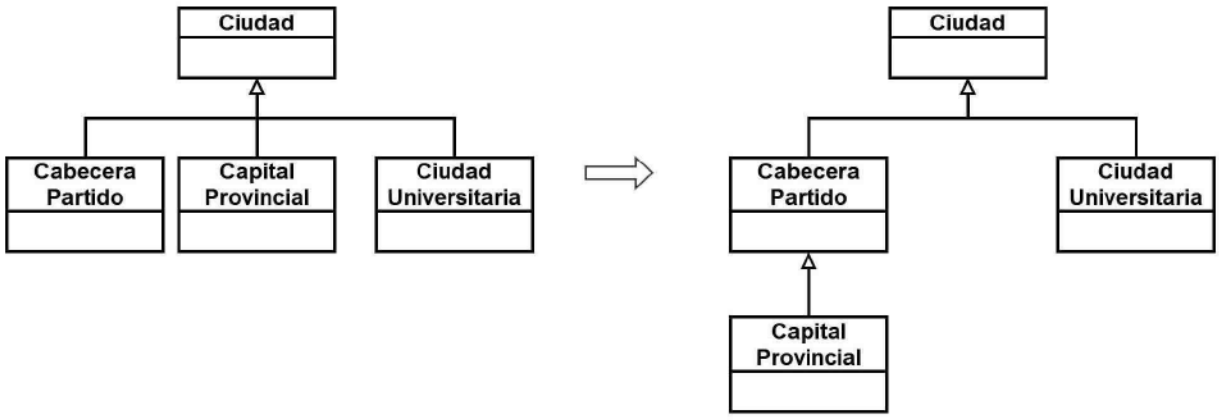


Imagen 19. Ejemplo.

**4. Delete subcategory relationship refactoring.**

**Descripción:**

Elimina una relación de subcategoría existente entre dos categorías. El objetivo de este ***refactoring*** es eliminar la relación de subcategoría existente.

En el estado inicial una categoría es subcategoría de otra. En el estado final las categorías no están relacionadas por la relación de subcategorias. Es posible que se requiera recategorizar algunos de los artículos de la subcategoria. Esta recategorización de artículos queda fuera de la responsabilidad del ***refactoring*** siendo el usuario el encargado de realizar las categorizaciones correspondientes.

**Parámetros:**

- Nombre de la categoría que es subcategoría.
- Nombre de la categoría que es supercategoría.

**Implementación:**

1. Eliminar de la página de la subcategoría la categorización de la supercategoría.

**Ejemplo:**

Siguiendo con el ejemplo anterior cuando se realiza la relación de subcategorización entre 'Capital Provincial' y 'Cabecera Partido' debemos eliminar la relación de subcategoría entre 'Capital Provincial' y 'Ciudad'. Se puede realizar una composición de ***refactorings*** utilizando este ***refactoring*** para implementar el anterior o también utilizar ***Delete category relationship refactoring*** para eliminar la relación de subcategorización.

**5. Change category refactoring.**

**Descripción:**

Dado un *artículo wiki* con una categorización, se elimina una categorización y se agrega una nueva. Este ***refactoring*** se lo puede ver como una composición de los ***refactorings*** **Delete category relationship Refactoring** y **Add Category relationship Refactoring**. El objetivo de este ***refactoring*** es agregar una categoría al conjunto de categorías que pertenece el artículo y eliminar una de las categorías existentes.

En el estado inicial existe un artículo con un conjunto de categorías a las que pertenece y una categoría que no pertenece al conjunto de categorías del artículo. En el estado final el artículo pertenece a una nueva categoría y no pertenece a una de las categorías que pertenecía en el estado inicial.

**Parámetros:**

- Título del artículo al cual se le actualiza la categoría.

- Nombre de la nueva categoría.
- Nombre de la categoría a eliminar.

**Implementación:**

1. Agregar la nueva categorización al artículo.
2. Eliminar la categorización de la subcategoría.

**Ejemplo:**

Supongamos que tenemos un artículo que pertenece a la categoría 'Cabecera Partido' pero también es 'Capital Provincial'. También suponemos que la categoría 'Capital Provincial' es subcategoría de 'Cabecera Partido'. En lugar de solamente agregar la categorización debemos mover el artículo de una categoría a otra.

**6. Hierarchy pull up category refactoring.**

**Descripción:**

Dada una categoría, sube un nivel en el árbol jerárquico de las categorías. El objetivo de este ***refactoring*** es lograr una categoría más genérica subiendo un nivel en el árbol de jerarquía.

En el estado inicial se cuenta con una categoría que cuenta con una supercategoría que a su vez tiene una supercategoría. Posiblemente se requiera la recategorización de artículos de la categoría que se está modificando, esta tarea le corresponde al usuario ya que se debe analizar cuales son los artículos que se deben recategorizar. Los artículos que deben ser recategorizados son aquellos que pertenecen a la categoría que se está modificando y también deben pertenecer a la supercategoría actual. La recategorización de artículos puede realizarse utilizando el ***refactoring Change category refactoring.***

Debido al sistema de categorización con herencia múltiple se requiere indicar por parámetro el título de la supercategoría actual y el título de la supercategoría nueva. De contar con un sistema con herencia simple, este ***refactoring*** se podría implementar sin la necesidad de solicitar el título del artículo de la supercategoría. Este ***refactoring*** se puede implementar mediante la composición de otros ***refactorings*** básicos. El objetivo es contar con un ***refactoring*** que implemente de forma directa y atómica estas acciones es brindar al usuario la capacidad de modificar jerarquías de categorías de forma sencilla.

En el estado final la categoría habrá subido un nivel siendo subcategoría de la supercategoría inicial.

**Parámetros:**

- Título del artículo de la categoría que se quiere subir.
- Título del artículo de la supercategoría actual.
- Título del artículo de la supercategoría nueva.

**Implementación:**

1. Agregar la nueva categorización a la categoría.
2. Eliminar la categorización anterior.

**Ejemplo:**

Supongamos que tenemos la categoría 'Capital Provincial' es subcategoría de 'Ciudad' y que la categoría 'Cabecera Partido' es subcategoría de 'Capital Provincial'. Luego, nos damos cuenta que todas las capitales provinciales son también cabecera de partido. Por lo cual, debemos definir 'Cabecera Partido' como subcategoría de 'Ciudad'. Se aplica este ***refactoring*** sobre la categoria 'Cabecera Partido' y sube un nivel en el árbol de categorías.

**7. Hierarchy pull down category refactoring.**

**Descripción:**

Dada una categoría, baja un nivel en el árbol jerárquico de las categorías. El objetivo de este ***refactoring*** es lograr una categoría mas específica bajando un nivel en el árbol de jerarquía.

En el estado inicial se cuenta con dos categoría que cuenta con una misma supercategoría. Se debe eliminar la categorización de la nueva supercategoría a aquellos artículos que pertenecen tanto a la categoría que se está modificando como a la nueva supercategoría. Este ***refactoring*** se puede implementar mediante la composición de otros ***refactorings*** básicos. El objetivo es contar con un ***refactoring*** que implemente de forma directa y atómica estas acciones es brindar al usuario la capacidad de modificar jerarquías de categorías de forma sencilla y automática. En el estado final la categoría habrá bajado un nivel siendo subcategoría de una categoría que antes estaba al mismo nivel.

**Parámetros:**

- Título del artículo de la categoría.
- Título del artículo de la supercategoria actual.
- Título del artículo de la supercategoria nueva.

**Implementación:**

1. Agregar la nueva categorización a la categoría.
2. Eliminar la categorización de la supercategoría actual.
3. De los artículos que pertenecen a la categoría que se modifica y también pertenecen a la nueva supercategoría, se debe eliminar la categorización de la nueva supercategoría.

**Ejemplo:**

Siguiendo con el ejemplo del ***refactoring*** anterior, ahora debemos hacer que la categoría 'Capital Provincial' descienda un nivel en la estructura jerárquica de las categorías. Luego, aplicando este ***refactoring*** se obtiene que 'Capital Provincial' es subcategoría de 'Cabecera Partido'.

**8. Join category refactoring.**

**Descripción:**

Dadas dos categorías se fusionan formando una sola. Todos los artículos de la segunda categoría pasan a la primera. El objetivo de este ***refactoring*** es fusionar dos categorías en una sola. Dadas dos categorías se fusionan en la primera mientras que la segunda es eliminada.

En el estado inicial existen dos categorías las cuales contienen artículos. Los artículos de la segunda categoría son agregados a la primer categoría. Posiblemente se requiera la categorización de artículos de la categoría que se está eliminando, esta tarea le corresponde al usuario ya que se debe analizar cuales son los artículos que se deben categorizar. Los artículos que deben ser categorizados son aquellos que pertenecen a la categoría que se elimina y deban pertenecer a alguna supercategoría de la categoría que se elimina. Debido a que este ***refactoring*** no modifica las supercategorías, ni las subcategorías de la categoría resultante (siguen siendo las supercategorías y subcategorías de la primer categoría). La categorización de artículos puede realizarse utilizando el ***refactoring Add category refactoring.*** En el estado final la primer categoría contiene los artículos de las dos categorías, mientras que la segunda categoría es eliminada.

**Parámetros:**

- Nombre de la categoría que sigue existiendo.
- Nombre de la categoría que es eliminada.

**Implementación:**

1. Agregar a la primer categoría los artículos de la segunda categoría.
2. Eliminar la categorización de los artículos de la segunda categoría.
3. Eliminar la segunda categoría.

**Ejemplo:**

Supongamos que tenemos las categorías 'Ciudad' y 'Urbe'. Luego nos damos cuenta que las dos categorías representan el mismo concepto. Por lo cual se decide unir las categorías. Una vez que se decide cual es nombre mas adecuado para la categoría se ejecuta el ***refactoring*** con los parámetros de forma adecuada.

**9. Split category refactoring.**

**Descripción:**

Dada una categoría se divide en dos categorías. Se crea una nueva categoría y un subconjunto de artículos de la categoría pasan a la nueva categoría. El objetivo de este ***refactoring*** es dividir una categoría en dos categorías creando una nueva. Los artículos que contiene la categoría original son recategorizados según corresponda.

En el estado inicial existe una categoría que posiblemente describa mas de un concepto real y se requiere dividir en dos categorías. Se debe proveer un criterio semántico de selección para los artículos que deben formar parte de la nueva categoría. En el estado final existen dos categorías cada una contiene un subconjunto del conjunto de artículos de la categoría inicial.

**Parámetros:**

- Nombre de la categoría a dividir.
- Nombre de la nueva categoría.
- Criterio semántico que agrupa al conjunto de artículos que se deben agregar a la nueva categoría.

**Implementación:**

1. Crear el artículo de la nueva categoría.
2. Agregar la categorización a los artículos que forman la nueva categoría.
3. Eliminar la categorización de la categoría que se divide.

**Ejemplo:**

Supongamos que dentro de la ***wiki*** contamos con una categoría llamada 'Ciudad'. Luego nos damos cuenta que la categoría contiene muchos artículos, con lo cual se analiza si se está describiendo mas de un concepto de categorización. Se encuentra un subconjunto de artículos que no son descriptos de forma adecuada por la categoría 'Ciudad'. Se divide la categoría con una nueva categoría llamada 'Asentamiento Rural'. Hasta el momento de la división de la categoría existían artículos que estaban incluidos en la categoría 'Ciudad' debido a que no existía una categoría mas específica que los describiera. Luego de la división los artículos que son asentamientos rurales y no ciudades pertenecen a la nueva categoría.

**Property refactoring.**

**10. Add subproperty refactoring.**

**Descripción:**

Dadas dos propiedades crea una relación de subpropiedad entre ellas. El objetivo de este ***refactoring*** es crear una relación de subpropiedad entre las dos propiedades indicadas.

En el estado inicial existen dos propiedades sin relación entre ellas. En el estado final existen dos propiedades donde una es subpropiedad de la otra.

**Parámetros:**

- Nombre de la propiedad que será como superpropiedad.
- Nombre de la propiedad que será subpropiedad.

**Implementación:**

1. Agregar la anotación [[subproperty of :: nombreSuperPropiedad]] al artículo de la subpropiedad.

**Ejemplo:**

Supongamos que tenemos una propiedad 'casado con' y también tenemos la propiedad 'felizmente casado con'. Luego, nos damos cuenta que todas las personas que están 'felizmente casado con' también están relacionados por la relación 'casado con'. Con lo cual se debe realizar la relación de subpropiedad. La propiedad 'felizmente casado con' es subpropiedad de 'casado con' ya que las personas que están felizmente casadas son un subconjunto de las personas que están casadas. De esta forma indicamos que cuando se consulta por las personas que están 'casado con' también se retornen las personas que están 'felizmente casado con'.

**11. Delete subproperty refactoring.**

**Descripción:**

Dadas dos propiedades, una subpropiedad de la otra, se elimina la relación de subpropiedad entre ellas. El objetivo de este ***refactoring*** es eliminar la relación de subpropiedad que existe entre las dos propiedades.

En el estado inicial existen dos propiedades siendo una subpropiedad de la otra. En el estado final existen dos propiedades sin ninguna relación entre ellas.

**Parámetros:**

- Nombre de la propiedad que es subpropiedad.
- Nombre de la propiedad que es superpropiedad.

**Implementación:**

1. Eliminar la relación de subpropiedad de la subpropiedad.

**Ejemplo:**

Supongamos que tenemos una propiedad 'casado con' que tiene una subpropiedad llamada 'concubinato con'. En un principio se puede pensar que las personas que están en concubinato también están casadas. Sin embargo, existe una diferencia legal entre las dos propiedades. Por lo cual, se decide eliminar la relación de subpropiedad ya que legalmente las personas en concubinato no están casadas.

### 12. Pull up property refactoring.

**Descripción**:

En una jerarquía de propiedades, la propiedad seleccionada sube un nivel en árbol de jerarquías. El objetivo de este ***refactoring*** es lograr una propiedad más genérica subiendo un nivel en el árbol de jerarquía.

En el estado inicial se cuenta con una propiedad que cuenta con una superpropiedad que a su vez tiene una superpropiedad. Debido al sistema de herencia múltiple en propiedades se requiere indicar por parámetro el nombre de la superpropiedad y el nombre de la nueva superpropiedad. De contar con un sistema de herencia simple, este ***refactoring*** se podría implementar sin la necesidad de solicitar el nombre de la superpropiedad. En el estado final la propiedad habrá subido un nivel siendo subpropiedad de la superpropiedad inicial.

**Parámetros**:

- Nombre de la propiedad que se quiere subir de nivel.
- Nombre de la superpropiedad.
- Nombre de la nueva superpropiedad.

**Implementación**:

1. Agregar la relación de subpropiedad a la propiedad a subir de nivel.
2. Eliminar la relación de subpropiedad de la superpropiedad anterior.

**Ejemplo:**

Supongamos que tenemos una propiedad llamada 'Pareja con' que cuenta con una subpropiedad llamada 'Casado con' que a su vez tiene una subpropiedad llamada 'Concubinato con'. Luego nos damos cuenta que las personas que están en concubinato no están casadas legalmente. Por lo cual decidimos que la propiedad 'Concubinato con' este al mismo nivel que 'Casado con' en el árbol de jerarquías. Con lo cual subimos un nivel y hacemos que la propiedad 'Concubinato con' sea una forma de pareja mas genérica.

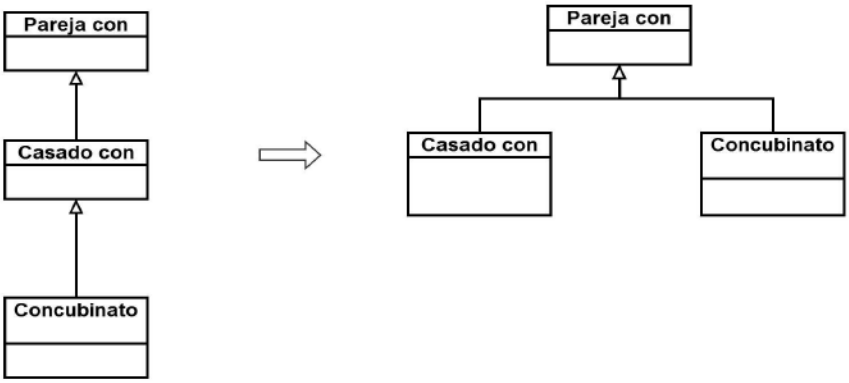


Imagen 20. Ejemplo.

### 13. Pull down property refactoring.

**Descripción:**

En una jerarquía de propiedades, la propiedad seleccionada se baja un nivel en el árbol de jerarquías. El objetivo de este ***refactoring*** es lograr una propiedad más específica subiendo un nivel en el árbol de jerarquía.

En el estado inicial se cuenta con una propiedad que cuenta con una superpropiedad que a su vez tiene una o varias subpropiedades. Debido al sistema de herencia múltiple en propiedades se requiere indicar por parámetro el nombre de la superpropiedad y el nombre de la nueva subpropiedad. De contar con un sistema de herencia simple, este ***refactoring*** se podría implementar sin la necesidad de solicitar el nombre de la superpropiedad y subpropiedad. En el estado final la propiedad habrá bajado un nivel siendo subpropiedad de una propiedad que estaba al mismo nivel en el árbol de jerarquía.

**Parámetros:**

- Nombre de la propiedad que se quiere bajar de nivel.
- Nombre de la superpropiedad.
- Nombre de la nueva superpropiedad.

**Implementación:**

1. Agregar la relación de subpropiedad a la propiedad a bajar de nivel.
2. Eliminar la relación de subpropiedad de la subpropiedad anterior.

**Ejemplo:**

Supongamos que tenemos una propiedad llamada 'Pareja con' que tiene como subpropiedad a 'Casado con', 'Concubinato con' y 'Felizmente Casado con'. Como se puede observar la propiedad 'Felizmente casado con' es mas específica que el resto de las subcategorías. Por lo cual se procede a bajar la propiedad un nivel en el árbol de jerarquía de propiedades. Como es de esperar se hace subpropiedad de 'Casado con'.

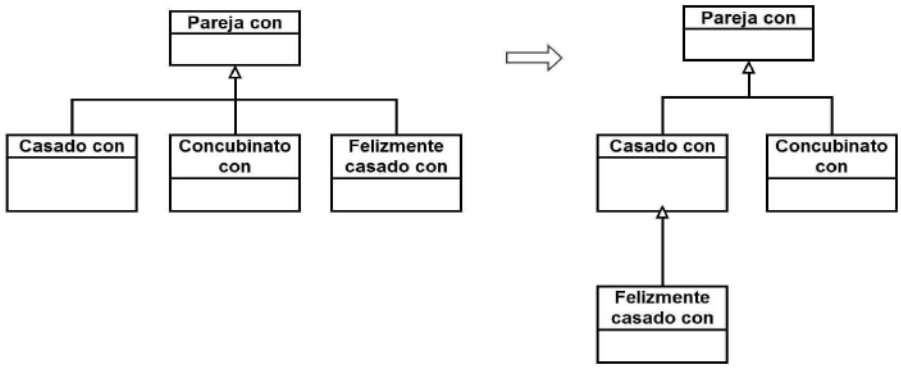


Imagen 21. Ejemplo.

**14. Join property refactoring**

**Descripción:**

Dadas dos propiedades se unen para formar una única propiedad. El objetivo de este ***refactoring*** es fusionar las dos propiedades en una sola.

En el estado inicial existen dos propiedades sin relación. Con el objetivo de unificar las propiedades se reemplazan las anotaciones de la propiedad que se elimina por anotaciones de la propiedad unificada. Se elimina el artículo de la propiedad eliminada ya que no cuenta con ninguna anotación. En el estado final se cuenta con una sola propiedad que representa a las dos propiedades unificadas.

**Parámetros:**

- Nombre de la propiedad sobre la cual se unifican.
- Nombre de la propiedad que se elimina.

**Implementación:**

1. Agregar las anotaciones sobre la cual se unifican.
2. Eliminar las anotaciones que se eliminan.
3. Eliminar el artículo de la anotación que se elimina.

**Ejemplo:**

Supongamos que tenemos una propiedad llamada 'Casado con' y otra propiedad llamada 'Matrimonio con'. Analizando los artículos donde son utilizadas las propiedades llegamos a la conclusión que las dos propiedades representan la misma relación entre personas. De esta forma se decide unificar las dos propiedades en una sola. Se unifican las dos propiedades en la propiedad 'Casado con' y se elimina la propiedad 'Matrimonio con'.

### 15. Split property refactoring

**Descripción:**

Dada una propiedad existente se divide para formar dos propiedades. Se crea una nueva propiedad. El objetivo de este ***refactoring*** es dividir una propiedad existente en dos propiedades, se crea una nueva propiedad y un subconjunto de las propiedades son reemplazadas.

En el estado inicial existe una propiedad que describe más de una relación real. Se debe proveer un criterio semántico de selección para los artículos en donde se debe reemplazar la propiedad original por la nueva propiedad. En los artículos que resultan seleccionados por el criterio semántico se reemplazan todas las ocurrencias de la propiedad original por la nueva propiedad. En el estado final existen dos propiedades donde las ocurrencias de la propiedad original están divididas entre la propiedad original y la nueva propiedad.

**Parámetros:**

- Nombre de la propiedad que se divide
- Nombre de la nueva propiedad.
- Criterio semántico que agrupa al conjunto de artículos que se deben agregar a la nueva categoría.

**Implementación:**

1. Crear un nuevo artículo para la nueva propiedad.
2. Reemplazar las anotaciones por la nueva propiedad.

**Ejemplo:**

Supongamos que tenemos una propiedad llamada 'Casado con' pero luego tenemos la necesidad de representar la relación de concubinato debido a que nos damos cuenta que la propiedad es utilizada para representar las personas que están en concubinato debido a que es una propiedad similar y la relación de 'Concubinato con' no existe. Dividiendo la propiedad 'Casado con' y creando la relación 'Concubinato con' se describe una relación de pareja nueva.

## 6. Conclusión.

La estrategia se basa en dos conceptos claves, los ***Semantic Wiki Bad Smells*** y en ***Semantic Wiki Refactorings***. El primer concepto es un síntoma en la estructura interna de la *ontología* que puede indicar la presencia de un error mas profundo. Mientras que el segundo concepto describe cuales son las acciones o procesos que debemos realizar para eliminar de forma total o parcial un ***bad smell***.

Un ***Semantic Wiki Bad Smells*** es un síntoma en la estructura semántica de una ***Wiki Semántica*** que posiblemente indique un problema mas profundo y sugiere que un ***refactorings*** debe ser aplicado. La aplicación de un ***refactoring*** resulta tan importante como determinar *cuando* es necesario aplicar ***refactorings*** y cuando dejar de hacerlo. La presencia de un ***Semantic Wiki Bad Smell*** no necesariamente indique un error, debe ser analizado en contexto para determinar si es necesario aplicar un ***refactoring***.

Un ***Semantic Wiki Refactoring*** representa el proceso necesario para eliminar total o parcialmente un ***bad smell*** con el objetivo de obtener una *ontología* de mejor calidad. Para la ejecución de los ***semantic refactorings*** es necesario la intervención del usuario debido a que ciertas modificaciones dependen del dominio y el contexto de la ***wiki semántica***.

El catálogo de ***semantic bad smell*** pretende detectar todos los problemas que son solucionados por los ***semantic wiki refactorings***. Cada uno de los ***semantic wiki bad smell*** detecta posibles errores de la ***ontología***, luego se indica cuales son los ***semantic wiki refactorings*** que pueden solucionar el problema. Se intenta encontrar mecanismos de detección automáticos, aunque muchas veces por la naturaleza de las ***wikis semánticas*** no es posible.

Se presenta el catálogo de ***semantic wiki refactorings*** el cual está expresado en terminología de ***Semantic Media Wiki (SMW)***. Si bien se presenta el contenido adaptado para ***SMW***, puede fácilmente ser modificado para otras ***wikis semánticas***. Muchas de los ***semantic refactorings*** pueden parecer muy básicos, pero realizando composiciones de ***refactorings*** se pueden generar modificaciones mas significantes.

Se presentó una solución que tiene como objetivo asistir al usuario en la continua mejoría de la *ontología* durante todas las etapas de la evolución. Se dan métodos para detectar posibles errores. Luego, se dan opciones para eliminar total o parcialmente el problema detectado utilizando ***semantic wiki refactorings***.

En el siguiente capítulo se presenta la implementación de la estrategia desarrollada. Se discuten los detalles de implementación y cuales fueron las decisiones de diseño que se tomaron. También se presenta una herramienta accesoria llamada ***Agrupador Semántico Web.***

# Capítulo 4

## Implementación

*Saludos a la bestia primitiva* !

*Y también al capitán Akab* !

***Alexander Supertramp.***

## Introducción.

En el siguiente capítulo se presenta las implementaciones realizadas para cumplir con los objetivos planteados. Se describen los análisis realizados y las decisiones de diseño que se tomaron. En primer lugar se describe la implementación de ***Semantic Wiki Refactorings***, luego de ***Semantic Wiki Bad Smells*** y finalmente se describe la herramienta ***Agrupador Semántico Web***.

Un ***Semantic Wiki Bad Smell*** describe un sintoma en la estructura de la ***Wiki Semántica***. Para detectar ese error es necesario seleccionar *artículos wikis* para determinar donde es necesario la aplicación de un ***refactoring.*** Para seleccionar artículos, en primer lugar se intento con las consultas semánticas, pero se encontró que la expresividad de las consultas no son suficientes para describir un ***bad smell***. Dado que las consultas semánticas no alcanzan, se provee un modelo que intenta brindar más información para la definición de consultas.

Un ***Semantic Wiki Refactoring*** provocar cambios en la estructura interna de una ***Wiki Semántica.*** Los cambios necesarios para realizar un ***refactoring*** se pueden descomponer en acciones básicas. Analizando los ***refactorings*** se obtuvo que muchas de las acciones básicas eran compartidas por los ***refactorings***. Se logra expresar todos los ***refactorings*** en término acciones básicas y realizar una implementación mas flexible y extensible.

Para la ejecución de los ***refactorings*** se utilizan consultas semánticas. Un usuario escribe una consulta semántica y luego los *artículos wiki* que resultan seleccionadas se les aplica un ***refactoring***. En el momento en el cuál se ejecutan los ***refactorings*** puede suceder que los artículos no contengan demasiada información semántica. Al no contar con suficiente información semántica los usuarios se ven imposibilitados de escribir una consulta semántica para agrupar el conjunto. Para solucionar este problema la herramienta ***Agrupador Semántico Web (Semantic Grouper)*** permite crear nuevos conjuntos semánticos de *artículos wiki* sin relación semántica entre ellas. Los usuarios crean nuevos conjuntos que luego son utilizados para la ejecución los ***refactorings***.

### 1. Semantic Wiki Bad Smells.

Un ***Semantic Wiki Bad Smell*** es un síntoma en la estructura semántica de la ***Semantic Wiki*** que posiblemente indique un error mas profundo y sugiere que un ***semantic refactoring*** debe ser aplicado. Se debe entender que la detección de un ***Bad Smell*** no necesariamente indica la presencia de un problema real. Una vez localizado el ***bad smell*** debe ser analizado en su contexto y analizar si es necesario la aplicación de un ***refactoring***.

Definir un ***bad smell*** ayuda a determinar cuando y donde aplicar un ***refactoring***. Para indicar donde es necesario aplicar un ***refactoring*** se debe seleccionar un *artículo wiki*. Un ***bad smell*** selecciona uno o varios *artículos wiki* donde se debe aplicar un ***refactoring***. El ***bad smell*** indica cuales son los ***refactorings*** que se pueden aplicar para eliminar el error detectado. De esta manera se ayuda al usuario a comprender cual es el síntoma que se quiere solucionar y también se le indica como eliminarlo.

La herramienta tiene como objetivo la asistencia al usuario de forma continua. Este objetivo genera la necesidad de tener, siempre que se pueda, la capacidad de detectar ***bad smells*** de forma automática. Se puede ver a los *artículos wiki* como los operandos de los ***refactorings***, por lo cual, saber donde aplicar un ***refactoring*** consiste en seleccionar un o más *artículos wiki* donde hacerlo. La forma de seleccionar artículos en **SMW** es a través de consultas semánticas. Los usuarios seleccionan *artículos wikis* describiendo una condición que involucra información semántica. La definición de un ***bad smell*** consiste en determinar cuales son las condiciones que debe cumplir un artículos para ser seleccionado. Las condiciones implican tanto datos semánticos de los artículos como del contexto donde se encuentra. Puede involucrar las categorías a las que pertenece o las propiedades que utiliza.

Algunos ***bad smell*** necesitan información del contexto del *artículo wiki*. Esta información muchas veces no es accesible desde una consulta semántica en **SMW**. Por esta razón se observa que las consultas semánticas no son suficiente para conceptualizar las consultas que necesita realizar un ***bad smell***. La capacidad de las consultas semánticas en **SMW** no es suficiente para definir los ***bad smell***. Se entiende la necesidad contar con mas información contextual para el análisis requerido por un ***bad smell***.

Se diseña un modelo para definir un ***bad smell*** como se ve en la siguiente imagen:

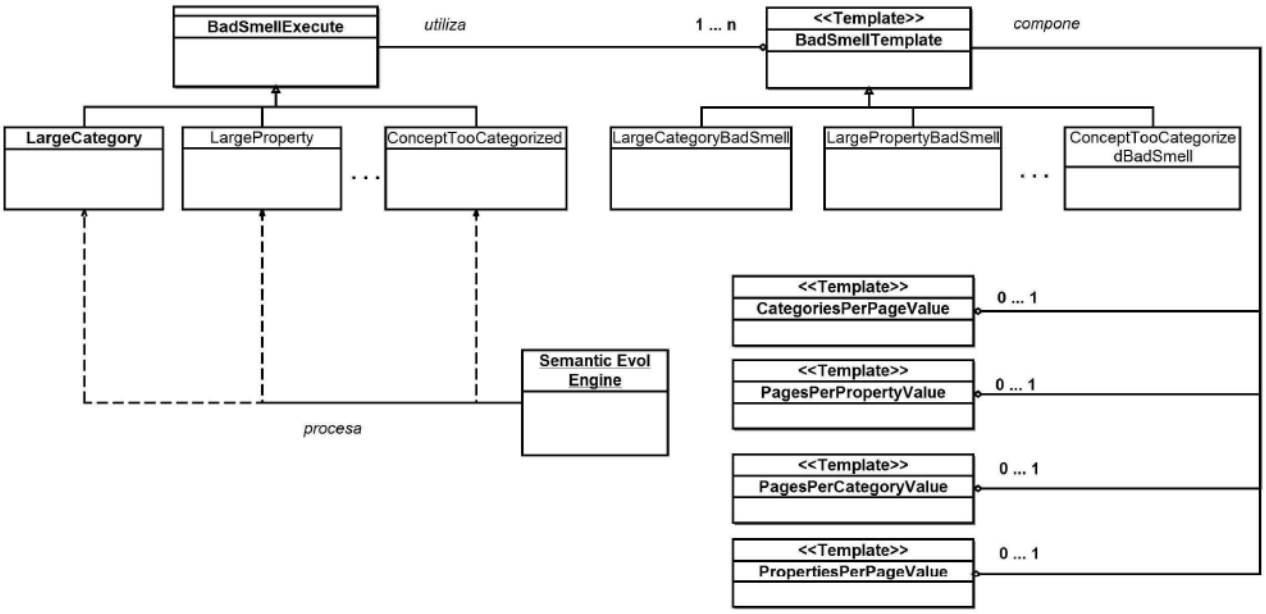


Imagen 22. Diagrama del modelo de Semantic Evol para ***bad smells***.

En la imagen vemos el modelo de **Semantic Evol** para ***bad smells***. Para definir un ***bad smell*** se debe definir un *template* dentro de la categoría ***BadSmellTemplate***. En el modelo vemos los ***bad smells*** *LargeCategoryBadSmell*, *LargePropertyBadSmell* y *ConceptTooCategorizedBadSmell*. Un ***BadSmellTemplate*** se *compone* de cero o un valor básico (***Basic Value***). Los valores básicos son cuatro y describen valores que no son posibles calcular con una consulta semántica. Por ejemplo, *CategoriesPerPageValue* es un valor que representa la cantidad de categorías a las que pertenece un *artículo wiki*, este valor no puede ser calculado por una consulta semántica. Luego, se define un artículo dentro de la categoría ***BadSmellExecute***. En el modelo vemos *LargeCategory*, *LargeProperty* y *ConceptTooCategorized*. Estos artículos utilizan uno o más ***BadSmellTemplate*** para definir una instancia de ejecución (definen el comportamiento del ***refactoring***). En la instancia de ejecución es donde el usuario define los parámetros con los que se ejecutará el ***bad smell***. Finalmente, se cuenta con ***Semantic Evol Engine*** que procesa un ***BadSmellExecute***. Realizar el procesamiento, consiste en calcular uno o mas ***Basic Value***, realizar la consulta que define el ***bad smell*** y retornar los artículos con los valores solicitados. A continuación se describen en detalle los elementos que componen el modelo de **Semantic Evol**.

**Basic Value.**

Para realizar la detección de ***bad smell*** se requiere información contextual que las consultas semánticas no proveen. Por lo cual se decidió que la herramienta realice los cálculos y estén disponibles para definir ***bad smells***. Los valores ayudan a decidir si se requiere la ejecución de un ***refactoring*** y son calculados para un artículo en particular.

**Categories Per Page Value.**

Se calcula la cantidad de categorías a las cuales pertenece un artículo. El sistema de categorías permite que un artículo pertenezca a más de una categoría. Si el artículo pertenece a muchas categorías, puede sugerir que el artículo esta describiendo más de un concepto real (*Concept too categorized Bad Smell*). Se tiene en cuenta la cantidad de categorías a las que pertenece de forma directa, no se tiene en cuenta las categorías a las cuales el artículo pertenece por herencia de categorías.

**Pages Per Property Value.**

Se calcula la cantidad de artículos en los cuales aparece una propiedad determinada. Una propiedad puede aparecer en muchos artículos. Si la propiedad está en demasiados artículos puede indicar que o bien, la propiedad no es entendida correctamente o que la propiedad esté representando más de una propiedad real (*Large Property)*.

**Pages Per Category Value.**

Se calcula la cantidad de artículos que tiene una categoría. Una categoría agrupa artículos con tópicos relacionados. Si la categoría contiene demasiados artículos, puede indicar que la categoría esta siendo mal utilizada o representa más de un tópico real (*Large Category Bad Smell)*.

**Properties Per Page Value.**

Se calcula la cantidad de propiedades que tiene un artículo. Dentro de un artículo puede haber más de una propiedad. Puede suceder que no haya ninguna propiedad (*Resource with no semantic annotation)* lo que puede indicar la necesidad de definir nuevas propiedades.

Cada uno de los ***Basic Values*** son *templates* dentro de la ***wiki***. Cuando un ***Bad Smell Template*** necesita uno de estos valores simplemente agrega el *template* a su artículo. El template de un ***Basic Value*** agrega información semántica para indicar al ***Semantic Evol Engine*** cual es el ***Basic Value*** requerido. En el momento de procesar el ***bad smell***, se utiliza la información semántica para determinar que valor se debe calcular. La herramienta realiza el cálculo del valor y luego utiliza ese valor para determinar que artículos se debe retornar.

En la siguiente imagen vemos la edición de uno de los *templates*. Vemos como se define la categoría a la cuál pertenece el *template,* pero vemos que esa categorización no está incluida en los artículos que utilizan el *template*. Luego, vemos como el *template* agrega una anotación semántica a aquellos artículos que lo utilizan. Cada ***Basic Value*** agregan la misma anotación con diferentes valores. De esta forma se indica al ***Semantic Evol Engine*** qué información debe calcular.

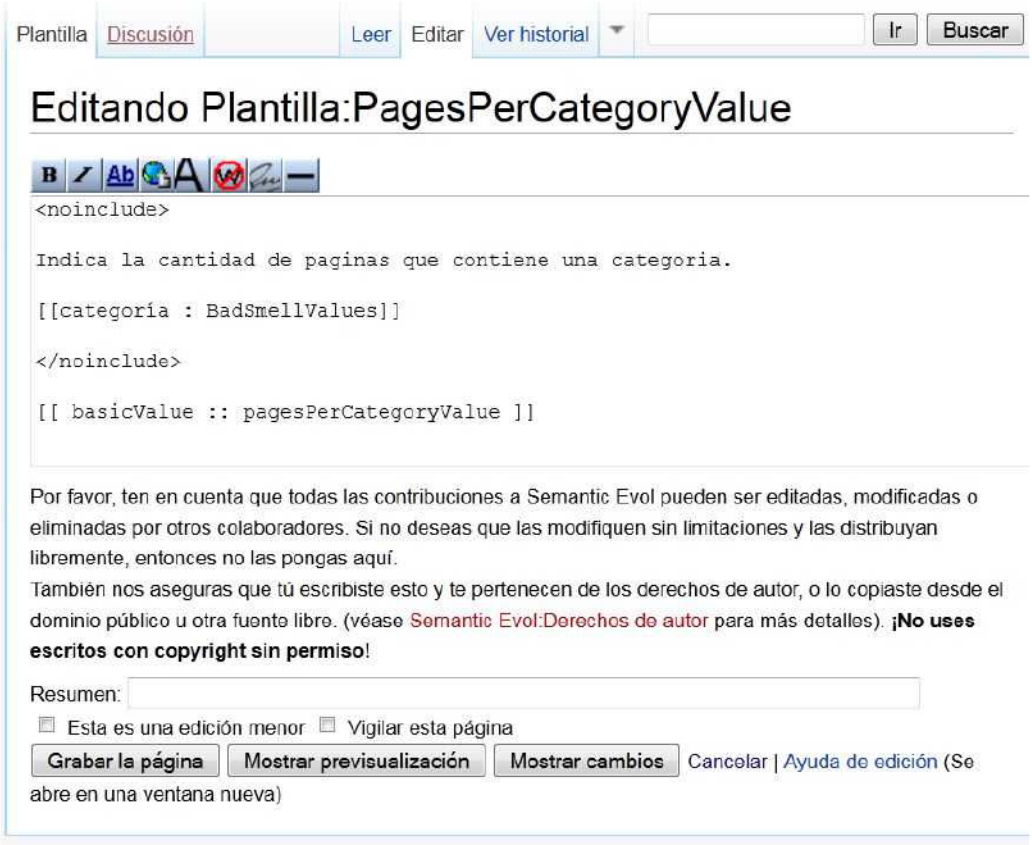


Imagen 23. Edición del template de un Basic Value.

**Bad Smell Template.**

Un ***Bad Smell Template*** define el comportamiento de un ***Bad Smell***. Define las anotaciones semánticas que serán utilizadas por el ***Semantic Evol Engine*** para procesar cada uno de los ***Bad Smell Execute***.

Se definen las siguientes anotaciones semánticas:

***Page Condition***: Es la condición de selección de *artículos wiki*. Para lograr una mayor expresividad y flexibilidad se provee la capacidad de analizar un conjunto de artículos definidos por una consulta. Los usuarios definen una consulta semántica que retornar *artículos wiki*, luego la herramienta analiza cada artículo y determina si debe incluirlo en la respuesta. Para esto se intentó utilizar una consulta semántica. Como se verá mas adelante, es necesario que la condición sea guardada como un valor de una anotación semántica. Existen ciertas limitaciones para los valores de las anotaciones y no se puede guardar una consulta como un valor. En este punto se decidió describir solamente la condición de la consulta, para luego a partir de esta información generar la consulta. Al intentar guardar solamente la condición se volvió a tener el mismo problema ya que no se pueden guardar ciertos caracteres especiales. Para solucionar este problema se decidió hacer un cambio de variables para poder expresar la parte de la condición dentro de un valor de una anotación semántica. Para poder definir una condición como valor de una anotación semántica se realiza un cambio de variables, intercambiando '[' por '/' , el carácter ']' por '\' y el carácter ':' por '$'.

Por esta razón si se intenta definir la siguiente consulta semántica :

```
{{#ask:  [[ Category:Hombre ]]
}}
```

El valor del parámetro condición debe ser:

```
//Category$Hombre\\
```

Luego la herramienta puede reconstruir la consulta semántica y seleccionar todos los artículos necesarios.

El separador se encuentra en las anotaciones semánticas también debe ser definido. Por ejemplo, en la anotación *[[ edad :: 9 ]]* el separador es '::' debido a que es una anotación de una propiedad. Las anotaciones de categorías tienen el símbolo ':' entre la palabra 'Category' y el nombre de la anotación, por ejemplo *[[ Category : Ciudad ]]*. Nuevamente se realiza un cambio de variables debido a que los caracteres '*::*' y '*:*' no pueden ser valores de una anotación. Por lo cual se utiliza el símbolo '*$$*' para indicar que se requiere el símbolo '*::*' y se utiliza el símbolo '*$*' para indicar que se requiere el símbolo '*:*'. Si bien se podría detectar cuando es necesario el símbolo '*:*' y cuando el símbolo '*::*' se intenta tener un mayor grado de libertad sobre las anotaciones que se quiere agregar.

***Basic Value Comparator***: Es un operador de comparación numérico que se expresa dentro de la anotación en formato de texto. Se lo utiliza para comparar un ***Basic Value*** calculado con un ***Limit Value***. Los cuatro valores posibles son: '*Higher*' que se interpreta como mayor '>' ; '*Less*' que se interpreta como menor '<' ; '*HigherEqual*' que se interpreta como mayor igual '>=' ; '*LessEqual*' que se interpreta como menor igual '<=' ; '*Equal*' que se interpreta como igual '='. Se cree que es mas natural para los usuarios expresar los comparadores en lenguaje natural y no en símbolos matemáticos.

***Limit Value***: es un valor límite que se utiliza para comparar un ***Basic Value*** calculado utilizando un ***Basic Value Comparator***. Si la comparación da verdadera, el artículo debe estar dentro del conjunto de respuesta del ***bad smell***. Establecer un valor límite ayuda a los usuarios a excluir aquellos artículos que no deben estar dentro de un ***bad smell***. Es decir, los usuarios establecen cuando una categoría es demasiado grande, luego se buscan todas las categorías que superen ese valor.

***Order***: define el orden en el que estarán listados los resultados. Los artículos que son resultado de un ***bad smell*** son listados en una tabla junto con un valor numérico. Se utiliza *order* para definir cual es el criterio de ordenación (tomando el valor numérico de cada artículo) para los artículos del resultado. Se brinda esta posibilidad dado que en algunos ***bad smells*** buscamos el valor máximo (del valor que nos devuelve la herramienta) por lo cual es deseable que ese valor este primero en la lista. Luego en otros ***bad smells*** buscamos el valor mínimo por lo cual es necesario que este en primer lugar los valores mínimos. Todos los artículos que califican para formar parte de la respuesta, son ordenados en una lista de acuerdo a su valor numérico, luego utilizando *order* definimos si nos interesan los artículos con valores mas grandes o los artículso con valores mínimos. Como se dijo, depende de la semántica que tenga el valor numérico y del ***bad smell*** que estemos definiendo serán los artículos que nos interesan. Si queremos un orden ascendente el valor debe ser '*up*' y si

queremos orden descendente el valor debe ser '*down*'.

***Limit***: define la cantidad máxima de artículos que se listan en el resultado. Podemos pedirle a la herramienta solo los diez primeros artículos que selecciona el ***bad smell***.

En la siguiente imagen vemos la edición de un ***Bad Smell Template***.


Imagen 24. Edición de un ***Bad Smell Template***.

En la imagen anterior se pueden ver las anotaciones que agrega el *template*. Junto con las anotaciones requeridas se agrega el *template* del *Basic Value* requerido. Las anotaciones agregan semántica a los parámetros del *template*, siendo así un *template* semántico.

**Bad Smell Execute.**

Un ***Bad Smell Execute*** utiliza un ***Bad Smell Template*** para definir una instancia de ejecución de un ***bad smell***. Los usuarios escriben un ***Bad Smell Execute*** para definir los parámetros de un ***bad smell.***

Los usuarios crean un nuevo artículo y utilizando un ***Bad Smell Template*** establecen cuales son los parámetros para la instancia de ejecución. Si se requiere crear otra instancia de ejecución se puede crear otro artículo o reutilizar el artículo. Si bien es posible reutilizar el artículo es interesante la creación de un artículo por instancia de ejecución ya que sirve de historial de ***bad smells*** que se crean en la ***Wiki***. Se deben definir los parámetros definidos por el ***Bad Smell Template*** que se desea utilizar.

En la siguiente imagen vemos la edición de un artículo que describe un ***Bad Smell Execute***:


Imagen 25. ***Bad Smell Execute.***

En la imagen anterior vemos como se utiliza el *template* con los parámetros deseados. Una vez que se tiene el artículos con los parámetros definidos, se guarda como cualquier otro artículo. Vemos en este ejemplo como se utiliza el cambio de símbolos para determinar cuál es la condición que deseamos para la consulta semántica requerida. En este caso el parámetro '**//$Category$+**' será interpretado como '**[[:Category:+]]**' con lo cual obtenemos todos los artículos de las categorías. Para ejecutar el ***Bad Smell Execute*** se debe acceder a la página principal de la herramienta.

Dependiendo del ***bad smell*** que se está ejecutando y del resultado que se obtiene, los usuarios analizan la respuesta y deciden que ***refactoring*** ejecutar. La página de un ***Bad Smell Execute*** describe cuales son los ***refactorings*** asociados al ***Bad Smell***.

**Semantic Evol Engine.**

Un ***Bad Smell Execute*** define los parámetros para los ***Bad Smell Template***. Un ***Bad Smell Template*** utiliza un ***Basic Value*** para definir un ***Bad Smell***. Finalmente se utiliza el ***Semantic Evol Engine*** que *procesa* un artículo ***Bad Smell Execute***. Con los valores de las anotaciones como parámetros devuelve un conjunto de artículos, luego dependiendo del ***Bad Smell*** que se trate los usuarios determinan donde es necesario la ejecución de un ***refactoring***.

La herramienta asiste al usuario en la toma de decisiones sobre donde y cuando ejecutar un ***refactoring***.

Dado que la decisión es contextual no se puede realizar de forma automática. Es el usuario el que debe determinar donde ejecutar un ***refactoring***. La herramienta brinda información sobre los ***bad smell***, cuales son los ***refactorings*** asociados y cuales son los síntomas que indican la presencia de un error. Luego de ejecutar un ***bad smell*** el usuario obtiene un conjunto de artículos que se deben analizar para determinar si existe el ***bad smell***.

La herramienta retorna al usuario un conjunto de artículos que son necesarios analizar para determinar si existe un ***bad smell***. Para seleccionar los artículos que se deben retornar, en primer lugar se seleccionan todas las páginas que la consulta semántica descripta por el usuario retorna. Luego para cada uno de los artículos se calcula un ***Basic Value***. Una vez que se tiene un ***Basic Value*** se realiza la comparación con el ***Basic Value Comparator***. En este punto se cuenta con dos valores numéricos y un comparador matemático. Para cada uno de los artículos, ***Semantic Evol Engine*** realiza la siguiente comparación:

```
(Basic Value) (Basic Value Comparator) (Limit Value)
```

Si la comparación es verdadera, el artículo es incluido en la respuesta de la herramienta para ser analizado por el usuario junto con el ***Basic Value*** calculado.

De esta forma el usuario obtiene un conjunto de artículos que deben ser analizados para determinar si existe un ***Bad Smell***. El ***Limit Value*** es provisto por el usuario debido a que es un valor que depende del contexto de la ***Wiki***. Por ejemplo, cuando se intenta encontrar las categorías que contienen muchos artículos, este valor depende de la cantidad de artículo que contiene la ***Wiki*** que también puede variar con el tiempo.

A continuación se muestra la página principal de la herramienta:

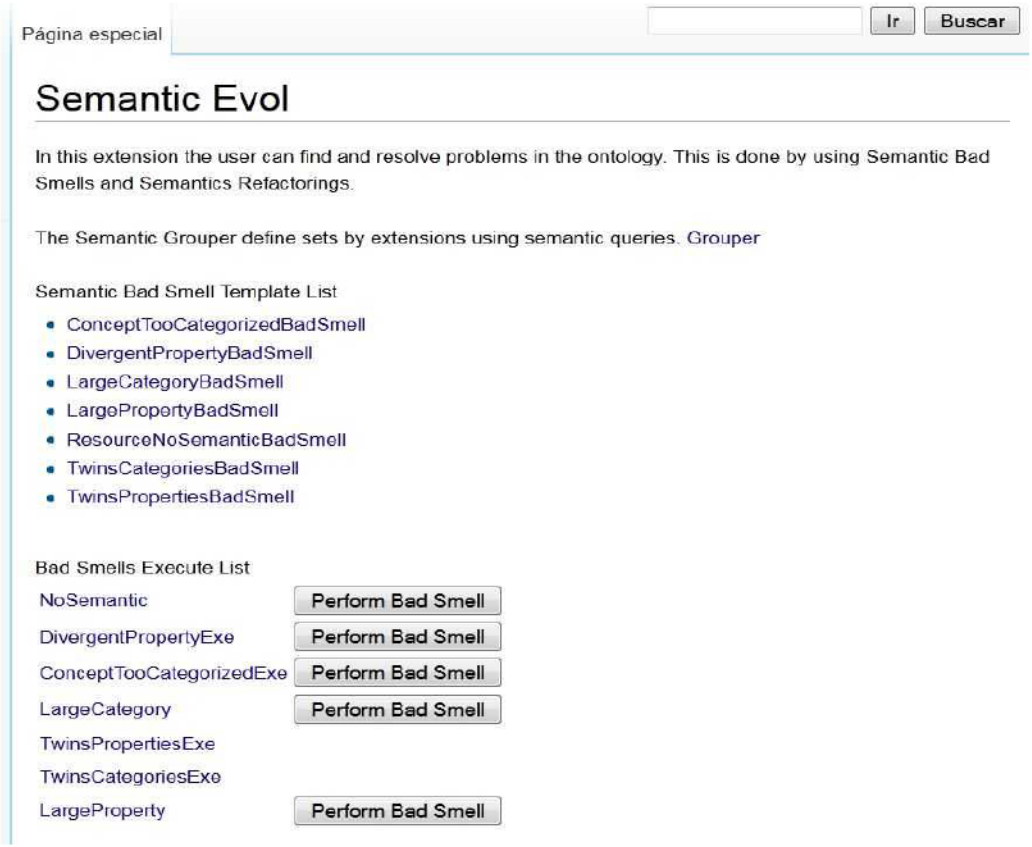


Imagen 26. Página principal de **Semantic Evol**.

Como vemos en la página principal de la herramienta se listan todos los artículos que pertenecen a la categoría *BadSmellExecute*. Para la ejecución, se dividen los ***bad smells*** en dos grupos. Aquellos ***bad smells*** que no tienen un botón para ejecutarse, son los que se pueden ejecutar con una consulta semántica. Luego, lo que cuentan con un botón para ejecutar (botón 'Perform Bad Smell') son los que necesitan del ***Semntic Evol Engine*** para ser ejecutados.

Los ***bad smells*** que no tienen un botón para ser ejecutados simplemente se accede al artículo y vemos si la consulta retorna algún resultado. En ese caso, se deberá analizar si se requiere ejecutar un ***refactoring*** para los artículos seleccionados.

En el caso de los ***bad smells*** que cuentan con un botón para ser ejecutados, se ejecutan dentro de la página de la herramienta simplemente accediendo al botón correspondiente. En la siguiente imagen vemos los resultados obtenidos al ejecutar un ***bad smell***:


Imagen 27. Ejecución de un ***bad smell***.

En la imagen anterior vemos los resultados de un ***bad smell***. Este ***bad smell*** (*Large Category*) se debe analizar si las categorías están bien utilizadas. Si existe una categoría que contiene demasiados artículos puede suceder que se esté entendiendo mal objetivo de la categoría.

En la imagen vemos cuales son los resultados del ***bad smell***. Como vemos en la imagen vemos que los artículos están ordenados de formas descendente y con un máximo de 15 artículos. Luego vemos que el valor limite de comparación es cero, por lo cual, se muestran incluso las categorías que tienen 1 artículo. Para una mejor comprensión vemos que las categorías con mas artículos están al principio, por lo cual, esos son las categorías que debemos analizar si es necesario la aplicación de un ***refactoring***.

**2. Semantic Wiki Refactorings.**

Utilizando ***Semantic Wiki Refactoring*** se provocan cambios en la estructura interna de una ***wiki semántica***. Los usuarios ejecutan los ***refactorings*** con el objetivo de eliminar total o parcialmente un ***Semantic Wiki Bad Smell***.

Se observa que los ***refactorings*** comparten modificaciones similares. Para lograr una implementación sencilla y clara se analiza una factorización el comportamiento. Se analiza cuál es el comportamiento común a los ***refactorings***. Para realizar este análisis se descomen el comportamiento en acciones básicas. Luego, se ve cuales de esas acciones básicas son compartidas por más de un ***refactoring***.

Los ***refactorings*** se descomponen para lograr una mejor calidad en la implementación. En esta etapa se tiene en cuenta que los ***refactorings*** iban a ser implementados en ***SMW***. La ventaja es que ***SMW*** guarda la información semántica dentro de los *artículos wikis*. Esto es una ventaja ya que para modificar la información semántica solo hay que modificar el contenido del *artículo wiki*.

Se encuentra una ventaja de implementación al tener la capacidad acceder a la estructura semántica de la ***wiki*** editando el contenido del *artículo wiki*. Se decide acceder a la información semántica a través de la modificación del contenido. Para realizar una modificación se simula una edición manual realizada por el usuario. De forma adicional, se obtiene la ventaja que la ***wiki*** registra todos los cambios (incluso intermedios) que realiza la herramienta. De esta forma los cambios intermedios realizados por los ***refactorings*** forman parte del historia del cambios de los artículos. La herramienta realiza ediciones de la misma forma que lo haría un usuario pero de forma automática.

Es necesario definir operaciones básicas para acceder a los *artículos wiki*. Se definen las siguientes acciones básicas: *add article refactoring* que crea un nuevo *artículo wiki* , *delete article refactoring* que elimina un *artículo wiki*, *add annotation refactoring que* agrega una anotación a un *artículo wiki*, *delete annotation refactoring* que eliminar una anotación de un *artículo wiki*.

A continuación se presenta el modelo de la herramienta **Semantic Evol** para ***refactorings:***

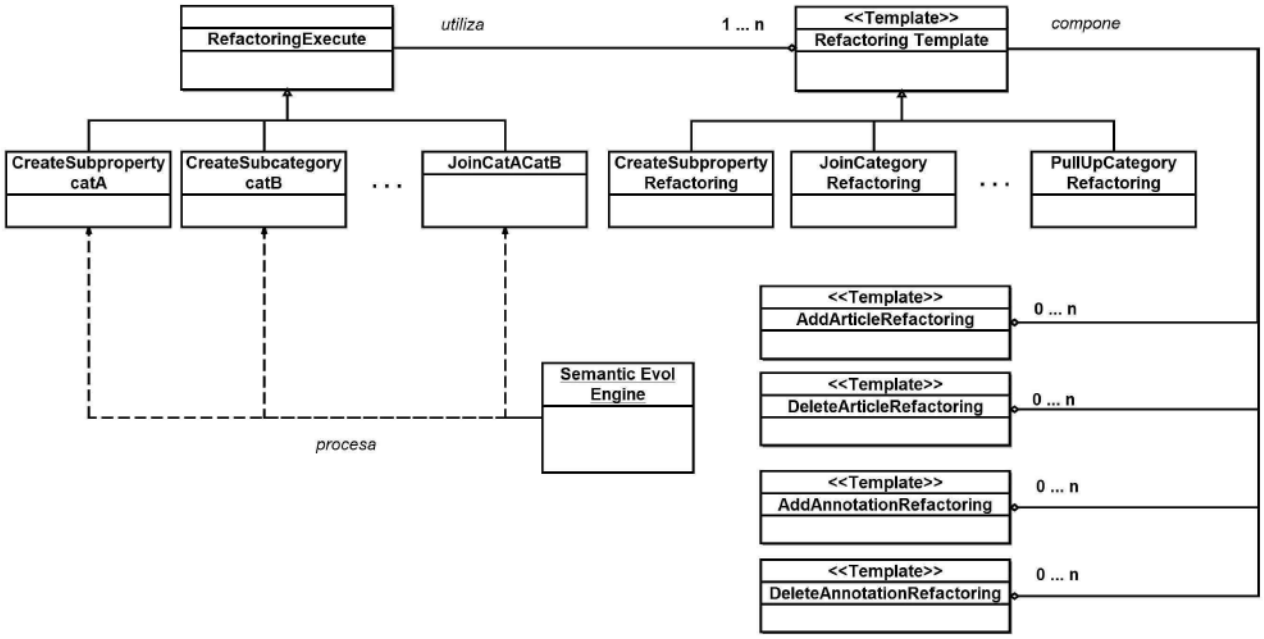


Imagen 28. Diagrama del modelo de Semantic Evol para ***refactorings***.

Un ***Refactoring Template*** se *compone* con cero o mas acciones básicas. Las acciones básicas que la herramienta provee son *AddArticleRefactoring*, *DeleteArticleRefactoring*, *AddAnnotationRefactoring*, *DeleteAnnotationRefactoring*. Para definir cuales son las acciones que debe hacer un ***refactoring*** se debe utilizar (como *templates)* cero o varias acciones básicas al *template* que define el ***refactoring***. Luego, los usuarios definen un ***Refactoring Execute*** donde se definen cuales son los parámetros con los que *utiliza* un ***Refactoring Template***. Finalmente, el ***Semantic Evol Engine*** *procesa* las instancias de ejecución y lleva a cabo las acciones básicas definidas en el ***Refactoring Template*** con los parámetros definidos en el ***Refactoring Execute***. A continuación se explica con mas detalles los componentes del modelo de **Semantic Evol** para ***refactorings***.

**Add Article Refactoring.**

Es un ***refactoring*** básico con el objetivo agregar un nuevo artículo a la ***wiki***. Se encontró que muchos ***refactorings*** tienen la necesidad de crear un nuevo artículo para definir un nuevo concepto.

Para agregar un nuevo artículo es necesario definir la siguiente información:

**Nombre** o **título** del nuevo artículo para agregar a la ***wiki***.

**Espacio de nombre** o *namespace* que define el espacio de nombre donde se quiere agregar al nuevo artículo.

Un **Id** que es un texto que se utiliza para identificar cuales son las tuplas que pertenecen a la misma instancia de ***refactoring***.

Los ***refactorings*** tienen la necesidad de agregar un nuevo artículo a la ***wiki***. Por ejemplo, cuando se

ejecuta el ***refactoring Split Category Refactoring*** es necesario crear un nuevo artículo para describir la nueva categoría que se crea con la ejecución del ***refactoring***.

**Delete Article Refactoring.**

Es un ***refactoring*** básico con el objetivo de eliminar artículos de la ***wiki***.

Para eliminar un artículo es necesario definir la siguiente información:

**Nombre** o **título** del artículo que se quiere eliminar de la ***wiki***.

Un **Id** que es un texto que se utiliza para identificar cuales son las tuplas que pertenecen a la misma instancia de ***refactoring***.

Los ***refactorings*** tienen la necesidad de eliminar artículos de la ***wiki***. Por ejemplo, cuando se ejecuta el ***refactoring Join Category Refactoring*** es necesario eliminar el artículo que describe la categoría que se deja de utilizar.

Un aspecto a tener en cuenta es que no se puede eliminar por completo un artículo. Si se elimina por completo un artículo luego un usuario puede volver a crearlo provocando el mismo error que dio origen a su eliminación. Por ejemplo, si se elimina una categoría en el momento que un usuario quiere utilizarla al no encontrarla puede volver a crearla.

Para evitar que los usuarios vuelvan a crear los artículos que la herramienta elimina, se realiza una eliminación lógica. Esto se lleva a cabo simplemente cambiando el contenido o texto del artículo. Modificando el texto se logra hacer una eliminación lógica. El artículo existe dentro de la ***wiki*** pero su contenido indica que no debe ser utilizado y se indica que fue eliminado por la herramienta.

Mediante la eliminación lógica se logra que los usuarios no vuelvan a crear los artículos que fueron eliminados. De esta manera se logra que el error que dio origen a la eliminación del artículo no vuelva a ser creado.

**Add Annotation Refactoring.**

Es un ***refactoring*** básico con el objetivo de agregar una anotación a un artículo.

Para agregar una anotación a un artículo es necesario definir la siguiente información:

El **nombre** de la anotación que se quiere agregar, es el lado izquierdo de la anotación semántica. Por ejemplo, si se quiere agregar la anotación *[[edad :: 9]]* el nombre de la anotación es '*edad*'.

El **valor** de la anotación que se quiere agregar, es el lado derecho de la anotación semántica. Por ejemplo, si se quiere agregar la anotación *[[edad :: 9]]* el valor de la anotación es '*9*'.

La **condición** de selección de los *artículos wikis* a las cuales se les va a agregar la anotación semántica. Se utiliza un cambio de variables, al igual que para las condiciones de ***Bad Smells Templates***. Con la condición se realiza una consulta que retorna artículos. A todos los artículos retornados por la consulta se les agrega la anotación indicada.

Un **separador** que define el símbolo que separa al nombre de la anotación del valor de la anotación. Es requerido debido a que el separador para una anotación definida por el usuario lleva el separador '::' y una anotación de categoría lleva el separador ':'. Por ejemplo, si se quiere agregar la anotación *[[edad :: 9]]* el separador debe ser '::'. Debido a las limitaciones antes mencionadas, se realiza un cambio de símbolos, por lo cuál si se requiere incluir el símbolo '::' se debe utilizar el símbolo '$$' y si se requiere el símbolo ':' se debe utilizar el símbolo '$'. Si bien se puede inferir cual de los dos separadores se requiere, se prefiere que lo ingrese el usuario para una mayor flexibilidad y extensibilidad.

Un **Id** que es un texto que se utiliza para identificar cuales son las tuplas que pertenecen a la misma instancia de ***refactoring***.

Existe un caso especial que debió ser tratado de forma diferencial. Algunos ***refactorings*** requieren agregar anotaciones con valores variables. Es decir, agregar el nombre de la anotación con diferentes valores en diferentes artículos. Para solucionar este requerimiento el valor de la anotación que queremos agregar debe ser el siguiente: 'SE_SameValueAs_NombreAnotación'. Si se agrega una anotación con este valor, lo que provoca es que la herramienta agrega una anotación con un nuevo valor. Este valor es calculado a partir del 'NombreAnotacion'. Se toma este nombre del texto y para cada *artículo wiki* que se agrega la anotación se obtiene el valor de la anotación 'NombreAnotacion' en ese artículo y es el valor que se utiliza para agregar la nueva anotación. Si la propiedad tiene mas de un valor en el artículo se toma el primer valor asignado a la propiedad. Eso es requerido por ejemplo en el caso de ***Join Property Refactoring*** donde la unión de dos propiedades requiere que la nueva propiedad unificada sea agregada a los artículos con el valor que tiene la propiedad que se elimina.

La ejecución del ***refactoring Add Annotation Refactoring*** genera que a todos los artículos seleccionados por la consulta semántica se les agrega la anotación formada por:

```
[[ nombreAnotación seperador valorAnotación ]]
```

Esta anotación es agregada al final del texto del artículo ***wiki***. Dado que la anotación que se agrega define estructuras semánticas, no se cree necesario definir un lugar específico dentro del texto del artículo. De ser requerida una posición el usuario deberá editar el artículo de forma manual.

A continuación vemos el lenguaje wiki del *template* Add Annotation:

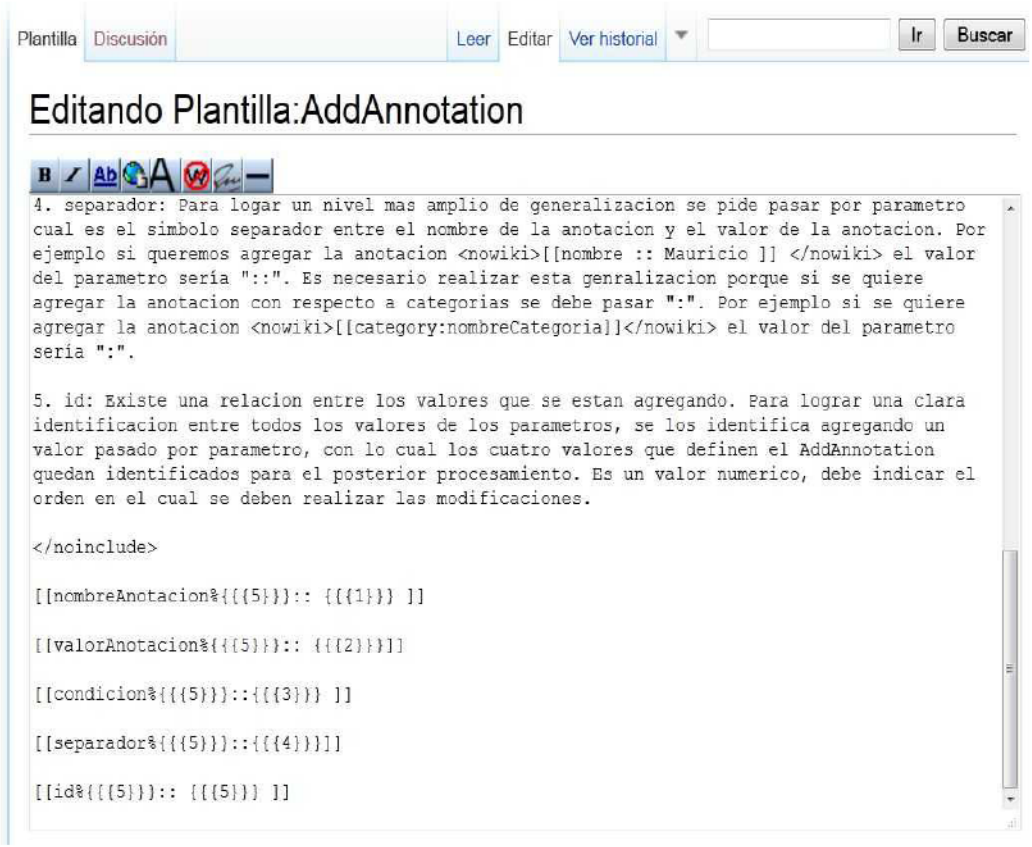


Imagen 29. Add Annotation.

En la imagen anterior vemos cuales son las anotaciones semánticas que agrega el *template* semántico. Dentro del nombre de las anotaciones semánticas se utiliza el símbolo '%' para lograr dividir el nombre de la anotación del Id del conjunto de anotaciones de un mismo ***refactoring***. Se realiza de esta forma para facilitar el procesamiento del ***refactoring***.

**Delete Annotation Refactoring.**

Es un ***refactoring*** básico que tiene como objetivo eliminar una anotación semántica de un artículo.

Para eliminar una anotación a un artículo es necesario definir la siguiente información:

El **nombre** de la anotación que se quiere eliminar, es el lado izquierdo de la anotación semántica. Por ejemplo, si se quiere eliminar la anotación *[[edad :: 9]]* el nombre de la anotación es '*edad*'.

El **valor** de la anotación que se quiere eliminar, es el lado derecho de la anotación semántica. Por ejemplo, si se quiere eliminar la anotación *[[edad :: 9]]* el valor de la anotación es '*9*'.

La **condición** de selección de los artículos a los cuales se les va a eliminar la anotación semántica. Se debe especificar la condición de la misma manera que en ***bad smells*** *template*.

El **separador** que se encuentra en la anotación semántica también debe ser definido. Se utiliza de la misma manera que en el ***refactoring*** básico ***Add Annotation Refactoring***.

Un **Id** que es un texto que se utiliza para identificar cuales son las tuplas que pertenecen a la misma instancia de ***refactoring***.

Existe el mismo caso que para ***Add Annotation Refactoring*** donde se debe eliminar una propiedad con un valor variable. Es decir, se debe eliminar la misma anotación con diferentes valores en diferentes artículos.

La ejecución del ***refactoring Delete Annotation Refactoring*** genera que a todos los artículos seleccionados por la consulta semántica se les elimina la anotación formada por:

*[[ nombreAnotacion separador valorAnotacion ]]*

La anotación es eliminada del texto solo si está contenida dentro del texto del contenido del artículo. Existe la posibilidad de que la anotación sea definida con un *template* semántico. En este caso la herramienta no eliminará la anotación ya que no se puede determinar el valor que tendrá la anotación que se quiere eliminar.

**Refactoring Templates.**

Un ***Refactoring Templates*** es un *template* semántico que define un conjunto ordenado de ***refactorings*** básicos que define el comportamiento de ***refactoring***. Son definidos por la herramienta y en el momento de la instalación y se importan como artículo.

Si bien los ***Refactoring Templates*** son definidos por la herramienta, los usuarios pueden definir nuevos ***Refactorings Templates***. Simplemente es un *template* que utiliza los ***refactorings*** básicos para definir el conjunto de acciones que definen su comportamiento. Una vez definido el comportamiento, el *template* debe pertenecer a la categoría ***Refactoring Templates***.

En la categoría ***Refactoring Templates*** se verán todas las implementaciones de los ***refactorings*** del catálogo. Definiendo nuevos ***refactorings*** los usuarios pueden extender el catálogo y hacer uso del modelo de ***Semantic Evol***.

A continuación vemos un ejemplo de un ***Refactoring Template***:

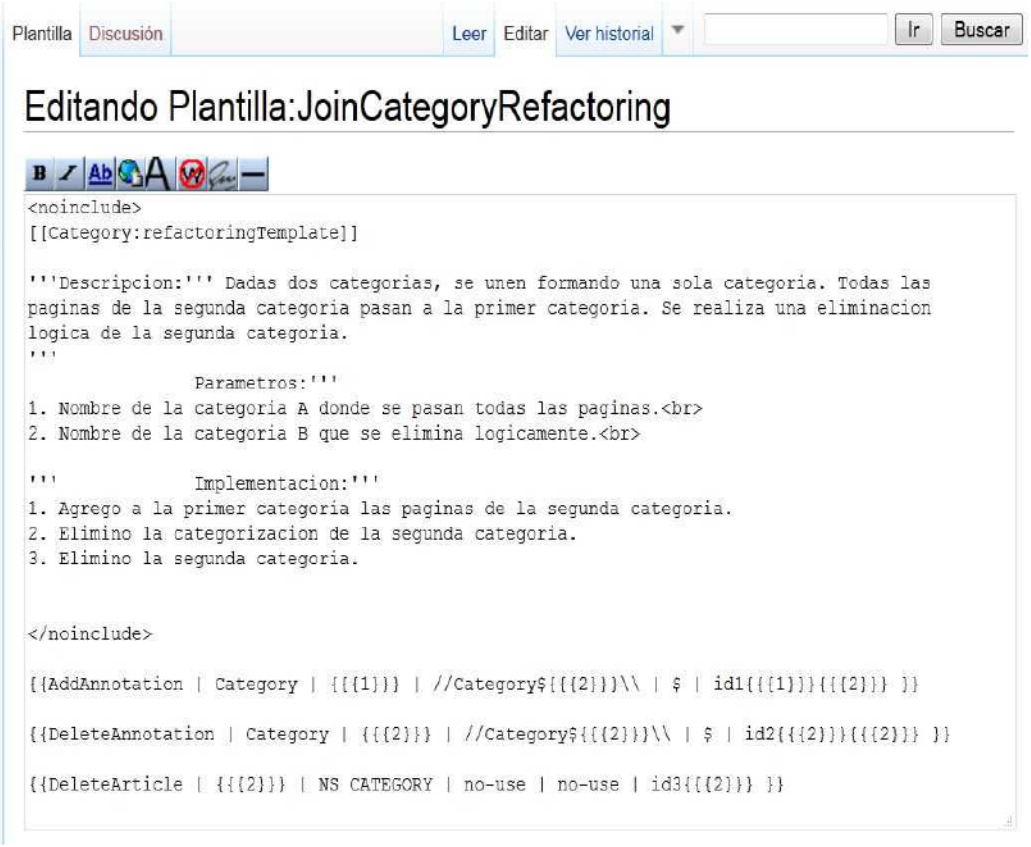


Imagen 30. ***Join Category Refactoring Template.***

En la imagen vemos como para definir el comportamiento de ***Join Category Refactoring*** se utilizan tres ***refactorings*** básicos. Primero se agrega una anotación a todos los artículos de la categoría que se elimina, indicando que ahora pertenecen a la primer categoría. Luego se elimina de los artículos que pertenecen a la categoría que se elimina la anotación que indica que pertenecen a esa categoría. Finalmente se elimina lógicamente el artículo de la categoría que se elimina.

**Refactoring Execute.**

Un ***Refactoring Execute*** es un artículo ***wiki*** que utiliza un ***Refactoring Template*** para definir una instancia de ejecución de un ***refactoring***. Para este fin se crear un *artículo wiki* que debe pertenecer a la categoría ***Refactoring Execute***. Para definir cual es el comportamiento de la instancia de ejecución el usuario utiliza uno o mas ***Refactoring Template*** definiendo los parámetros que sean necesarios.

El usuario deberá indicar cual es el estado de la instancia de ejecución. Se cuenta con dos estados posibles, *toExecute* y *executed*. Si la instancia está en el estado *toExecute*, se indica que está lista para ser ejecutada. En la página principal de la herramienta ***Semantic Evol*** se cuenta con un listado de todas las instancias que están listas para ser ejecutadas. Desde esta página los usuarios podrán ejecutar los ***Refactoring Execute***.

Un vez que la herramienta ejecuta un ***Refactoring Execute*** cambia su estado a *executed*. De esta manera se indica que la instancia ya fue ejecutada. Luego la instancia no aparece en la página principal, dado que ya fue ejecutada. Si bien se cambia el estado de la instancia de ejecución, el artículo ***wiki*** sigue existiendo y puede ser utilizada como documentación para trazar un historial de modificaciones en la ***wiki***.


Imagen 31. ***Join Category Execute.***

En la imagen vemos como se utiliza el *template JoinCategoryRefactoring*. En este caso el usuario desea unir las categorías 'joinA' y 'joinB'. Luego de la ejecución del ***refactoring*** la categoría 'joinB' estará eliminada lógicamente. También vemos como el usuario indica el estado deseado. Indicando el estado *toExecute* se le indica a la herramienta que todavía no fue ejecutado y se debe agregar a la lista de ***refactorings*** para ser ejecutados en la página principal de la herramienta.

**Semantic Evol Engine.**

Los ***Refactorings Execute*** utilizan los ***Refactorings Templates*** para definir una instancia de ejecución. Los ***Refactorings Templates*** utilizan los ***Refactorings*** básicos para definir su comportamiento. Los ***refactorings*** básicos son los únicos *templates* semánticos del modelo que agregan semántica. Se utiliza el ***Semantic Evol Engine*** para procesar los artículos ***Refactoring Execute*** y utilizando la información semántica que contienen se realizan las modificaciones necesarias.

Utilizando la información semántica se realizan las modificaciones. Cada uno de los ***refactorings*** básicos especifica información semántica. Luego, la herramienta con esta información y los valores de las anotaciones realiza diferentes modificaciones para diferentes ***refactorings*** básicos.

En la siguiente imagen vemos el listado de los ***Refactoring Templates*** con los que se cuenta y los ***Refactoring Execute*** que están para ser ejecutados (en estado *toExecute).*


Imagen 32. ***Refactorings*** a ser ejecutado.

En la imagen anterior vemos como un ***Refactoring Execute*** cuyo estado es 'toExecute' es listado en la pagina principal de la herramienta. Para proceder a la ejecución debemos acceder al botón '***Execute Refactoring***'.

En la siguiente imagen vemos la página de la categoría JoinA antes de la ejecución del ***refactoring***:

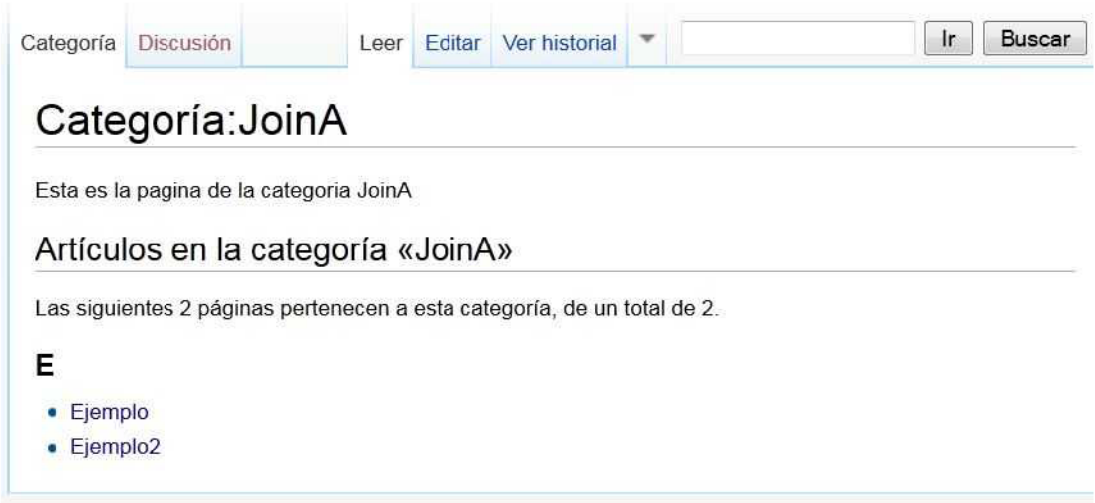


Imagen 33. Categoría JoinA antes de la ejecución del ***refactoring***.

Vemos que la categoría JoinA contiene dos artículos llamados 'Ejemplo' y 'Ejemplo2'.

En la siguiente imagen vemos la pagina de la categoría JoinB antes de la ejecución del ***refactoring***:


Imagen 34. Categoría JoinB antes de la ejecución del ***refactoring***.

Vemos que la categoría JoinB tiene dos artículos llamados 'Ejemplo3' y 'Ejemplo4'. En la siguiente imagen vemos al artículo 'Ejemplo3' antes de la ejecución.

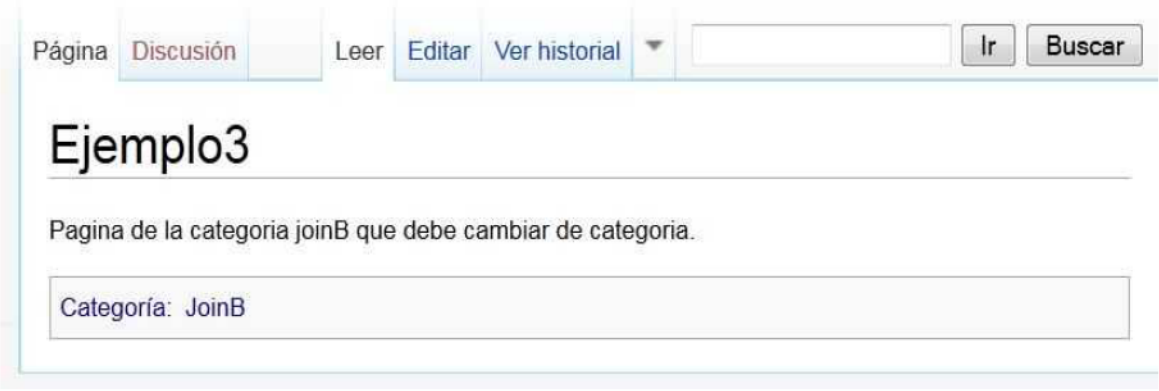


Imagen 35. Artículo 'Ejemplo3' antes de la ejecución del ***refactoring***.

Vemos que el artículo pertenece a la categoría JoinB.

En la siguiente imagen vemos como la herramienta ejecuta un ***refactoring***:


Imagen 36. Resultado de ejecutar un ***refactoring***.

En la imagen anterior vemos cuales son todas las modificaciones que realizó el ***refactoring***. En primer lugar nos indica cual es el título de la página que se afecta, la anotación con la cual se realizó y la acción realizada. Podemos ver que el ***refactoring*** realizó cinco modificaciones en la *ontología*.

Agrego la anotación '[[Category:JoinA]]' al artículo con título 'Ejemplo3'.

Agrego la anotación '[[Category:JoinA]]' al artículo con título 'Ejemplo4'.

Eliminó la anotación '[[Category:JoinB]]' al artículo con título 'Ejemplo3'.

Eliminó la anotación '[[Category:JoinB]]' al artículo con título 'Ejemplo'.

Eliminó el artículo con título 'JoinB' del espacio de nombres 'Category'.

A continuación vemos la página de la categoría JoinB después de la ejecución del ***refactoring***.


Imagen 37. Categoría JoinB después de la ejecución del ***refactoring***.

Como podemos ver **Semantic Evol** realiza una eliminación lógica modificando el contenido e indicando que la página fue eliminada y no debe ser utilizada.

A continuación vemos la página de la categoría JoinA después de la ejecución del ***refactoring***.


Imagen 38. Categoría JoinA después de la ejecución.

Como podemos ver, ahora la categoría JoinA contiene los artículos que tenía junto con los artículos de la categoría JoinB.

A continuación vemos el artículo Ejemplo3 después de la ejecución del ***refactoring***.

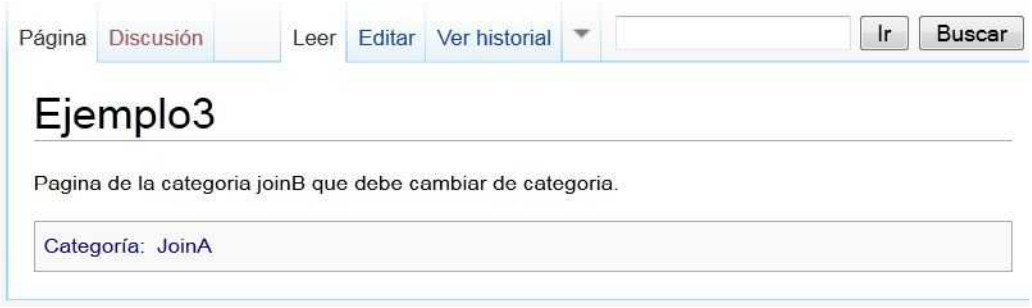


Imagen 39. El artículo Ejemplo3 después de la ejecución del ***refactoring***.

Como podemos ver, el artículo Ejemplo3 ahora es de la categoría JoinA.

### 3. Agrupado Semántico Web (*Semantic Grouper*).

Dentro de **SMW** las consultas semánticas se utilizan para seleccionar artículos. Los usuarios son capaces de seleccionar artículos que cumplan con una determinada condición semántica. Un usuario escribe una consulta semántica, define cual es la condición de selección y luego al procesar la consulta **SMW** retorna las páginas que cumplen con la condición. Por lo general, las condiciones se basan en la información semántica de los *artículos wiki* aunque un artículo pueda ser seleccionado simplemente por su título.

Dentro de la extensión **Semantic Evol**, los usuarios ejecutan los ***refactorings*** seleccionando los *artículos wiki* que intervienen en ellos con consultas semánticas. Las consultas semánticas son muy expresivas y pueden seleccionar artículos por diferentes criterios o componer resultados de varias consultas o subconsultas. El momento en el cual los usuarios requieren esta selección es cuando la *ontología* está evolucionando. Puede suceder que los artículos no contengan un criterio de selección posible y los usuarios no sean capaces de escribir una condición que logre seleccionar los artículos deseados.

Las consultas semánticas propuestas por **SMW** no son suficiente para la selección de páginas. Los usuarios deben ser capaz de agrupar artículos con criterios no establecidos por las características o anotaciones semánticas. Esta herramienta intenta solucionar el problema de la selección de *artículos wiki* que no contengan criterios de selecciones comunes. Los usuarios serán los encargados de seleccionar manualmente las páginas que requieran permanecer a un mismo conjunto.

Debe ser posible la creación de conjuntos de artículos. Un conjunto es formado por la sucesiva selección de artículos de forma manual. Para esta tarea, la herramienta provee a los usuarios un campo para ingresar una consulta semántica, luego se muestran todos los artículos resultantes de la consulta y sobre este conjunto seleccionan cuales formarán parte del nuevo conjunto.

En la siguiente imagen vemos como se utiliza el Agrupador Semántico Web:

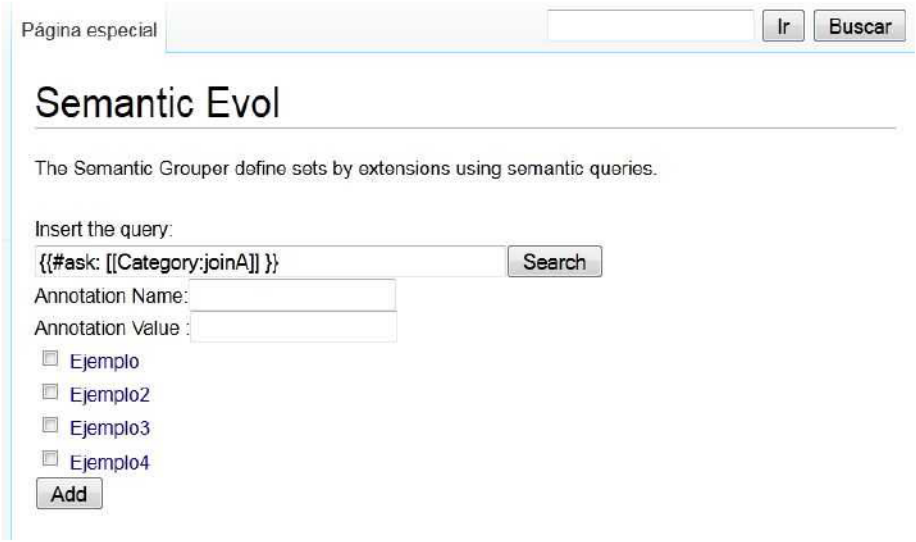


Imagen 40. ***Semantic Grouper.***

En la imagen anterior vemos como el usuario escribe una consulta, luego realiza la búsqueda y la herramienta retorna todos los artículos que resultan seleccionados por la consulta. El usuario selecciona solo aquellos artículos que desea que formen parte del nuevo conjunto.

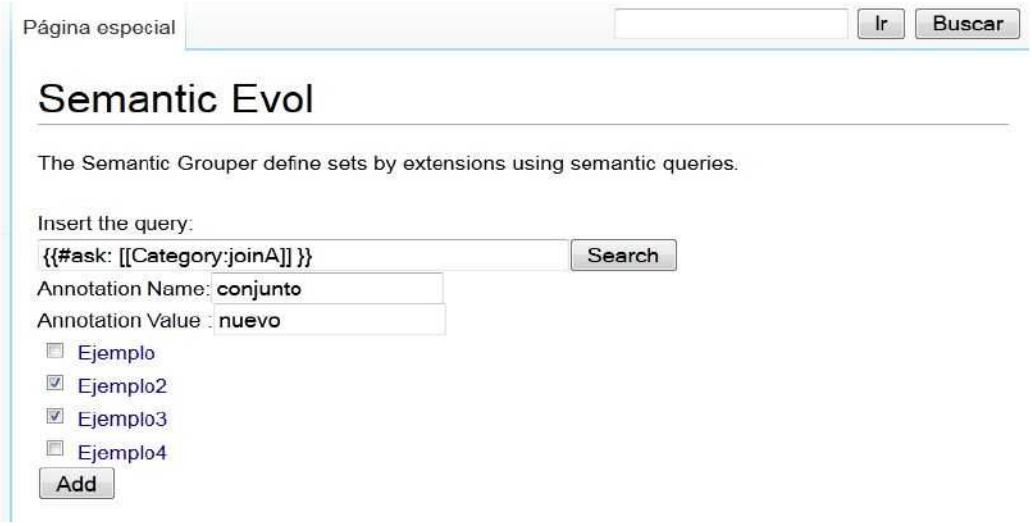


Imagen 41. El usuario selecciona los artículos.

El usuario selecciona aquellos artículos que desea que sean parte del conjunto. Luego en el campo *Annotation Name* ingresa el nombre de la anotación, y en *Annotation Value* el valor que deberá tener la anotación. Luego hacemos *Add* para agregar la anotación a los artículos seleccionados. Este proceso de escribir una consulta, ver cuales son los resultados obtenidos y luego seleccionar solo algunos artículos se puede repetir las veces que sea necesario. En este aspecto se dice que la creación del conjunto es por extensión, el usuario selecciona uno a uno los artículos que pertenecen al nuevo conjunto.

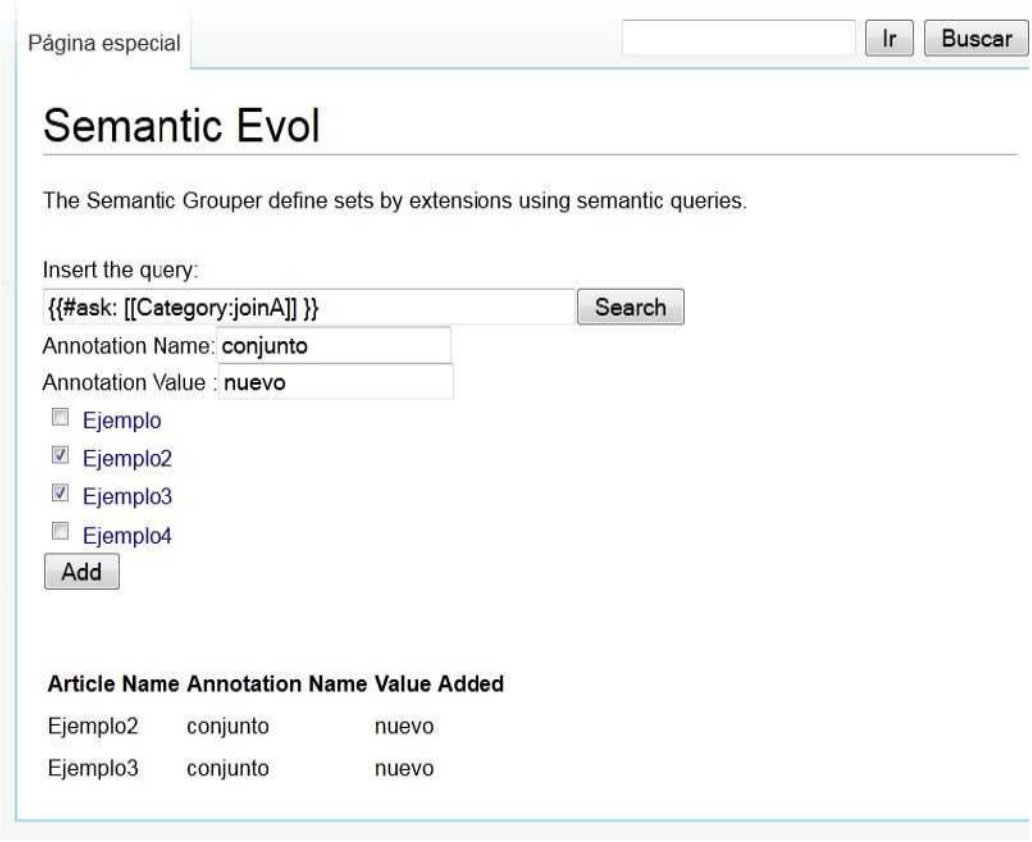


Imagen 42. Resultado de agregar una anotación a un conjunto de artículos.

Cuando el usuario decide agregar la anotación a los artículos la herramienta muestra en un formato de tabla cuales fueron las anotaciones que se agregaron y el nombre o título del artículo al cuál se agrego.

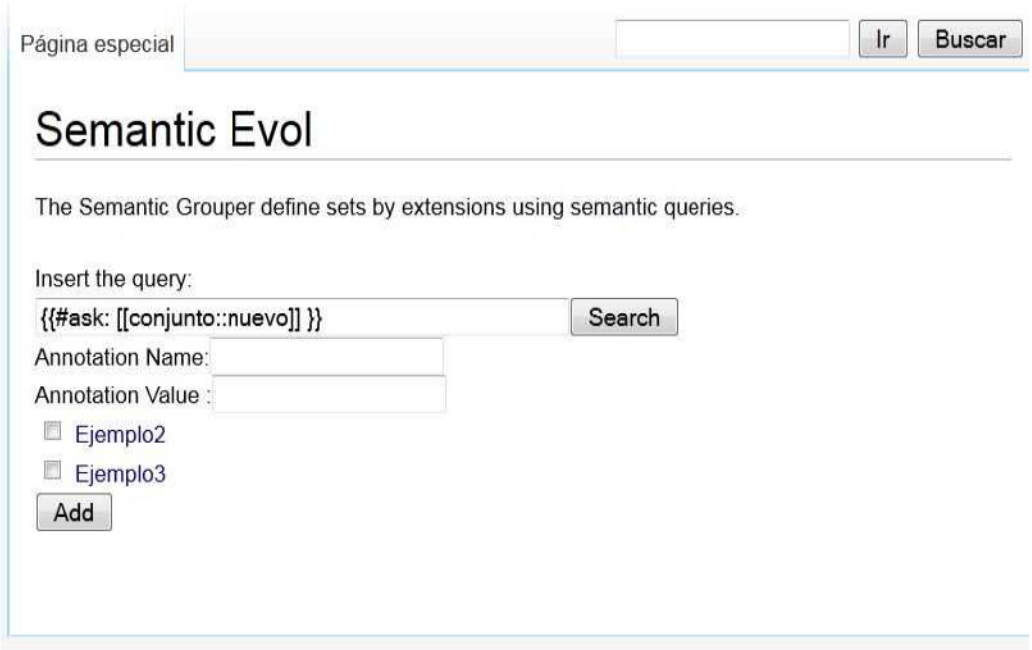


Imagen 43. Nuevo conjunto.

Finalmente el usuario puede escribir una consulta semántica que retorne solo aquellos artículos que

desea.

La creación del nuevo conjunto se logra agregando información semántica dentro de los artículos. La herramienta no permite la categorización de artículos de forma masiva. El objetivo de la herramienta es agregar información semántica para que sea posible seleccionar artículos por medio de una única consulta semántica. No se tiene como objetivo la categorización masiva de artículos. A partir de este momento se puede escribir una sencilla consulta semántica que permita seleccionar el nuevo conjunto.

Los usuarios pueden definir un nuevo conjunto donde sea posible seleccionar las páginas con una consulta semántica. Esta consulta es la que utilizan los usuarios para definir las páginas que deben ser modificados por un ***refactoring***.

Teniendo una herramienta para formar nuevos conjuntos semánticos de artículos se provee mas flexibilidad en el momento de ejecutar un ***refactoring***. Los usuarios pueden seleccionar artículos que no contengan criterios semánticos comunes para luego aplicar un ***refactoring***.

## 4. Conclusión.

Se describe la implementación de los ***Semantic Wiki Bad Smells*** y ***Semantic Wiki Refactoring***. Para cada uno de ellos de discutieron las decisiones de diseño tomadas en la implementación para cumplir con los objetivos planteados.

En la implementación de los ***Semantic Wiki Bad Smells*** el modelo muestra como la herramienta provee cuatro valores básicos, llamados ***Basic Values*** para definir ***bad smells***. Se cuenta con *templates* semánticos, llamados ***Bad Smells Template***, que definen el comportamiento de un ***bad smells***. Estos *templates componen* diferentes ***Basic Values***. Luego, los usuarios crean artículos donde se definen cuales son los parámetros con los que se utilizan los ***Bad Smell Templates***. Los artículos que definen los parámetros son llamados ***Bad Smell Execute***. Finalmente la herramienta utiliza el ***Semantic Evol Engine*** para *procesar* las instancias de ***Bad Smell Execute***.

En la implementación de los ***Semantic Wiki Refactorings*** el modelo muestra como la herramienta provee cuatro acciones básicas para definir ***refactorings.*** Las cuatro acciones básicas que se provee son *AddArticleRefactoring*, que agrega un artículo nuevo a la ***wiki***, *DeleteArticelRefactoring*, que elimina un artículo de la ***wiki***, *AddAnnotationRefactoring* que agrega una anotación a un artículo y *DeleteAnnotationRefactoring* que elimina una anotación de un artículo. Luego, se definen *templates* que componen acciones básicas. Cada *template*, llamados ***Refactoring Template***, puede utilizar cero o más acciones básicas para definir el comportamiento de un ***refactoring***. Los usuarios definen artículos llamados ***Refactoring Execute*** donde definen cuáles son los parámetros con los que se ejecutaran los ***refactorings***. Finalmente, el ***Semantic Evol Engine*** *procesa* los ***Refactoring Execute***, realizando las acciones necesarias dentro de la *ontología* de la ***wiki***.

La herramienta ***Agrupador Semantic Web (Semantic Grouper)*** brinda a los usuarios la posibilidad de crear nuevos conjuntos semánticos. Los usuarios definen una consulta semántica y una anotación. Luego, la herramienta muestra los resultados de la consulta. De todas las páginas que forman parte del resultado de la consulta, los usuarios seleccionan aquellas que formarán el nuevo conjunto. Luego, la herramienta agrega la anotación semántica a todos los artículos que fueron seleccionados. El proceso puede continuar definiendo una nueva consulta y agregando la misma anotación a otras páginas.

En este capítulo se vieron cuales son las decisiones de diseño que se tomaron en el momento de la implementación. Se presento el modelo de ***Semantic Evol*** para la implementación de los ***Semantic Wiki Refactorings*** y ***Semantic Wiki Bad Smell***. Para concluir se presento la herramienta ***Agrupador Semántico Web*** que permite la creación de nuevos conjuntos semánticos. En el siguiente capítulo se explican cuales son los aportes realizados por esta tesis y el trabajo futuro propuesto.

# Capítulo 5

## Conclusiones

*No se puede cambiar de corazón como de sombrero, sin haber sufrido primero.*

***Andrés Calamaro.***

## 1. Aportes realizados.

En una ***Wiki Semántica*** se requiere que los usuarios generen información semántica para el contenido. El principal aporte realizado es mejorar la experiencia de los usuarios en la utilización de tecnologías semánticas. Al experimentar una mejor experiencia utilizando una ***Wiki Semántica*** se espera que los usuarios generen más información semántica y de mejor calidad. La herramienta **Semantic Evol** mejora la calidad de la *ontología* en las ***Wikis Semánticas***. Mejorando la calidad de la *ontología* se mejora la navegabilidad, la exactitud de las consultas semánticas y la usabilidad de la ***Wiki***.

En la actualidad la mayor parte de la información semántica disponible se genera de forma automática. Utilizando herramienta como *templates* semánticos o extensiones de ***Wikis Semánticas*** que generan información semántica a partir del ingreso de datos por parte del usuario. Por ejemplo, *Semantic Forms* [8] genera información semántica a partir de formularios *html*. Con el objetivo de promover las tecnologías semánticas se aporta una estrategia para realizar el mantenimiento necesario de la información semántica disponible. Puede suceder que el contexto se modifique y se requieran modificaciones en los contenidos semánticos disponibles, con esta estrategia se brinda la posibilidad de analizar y evaluar la posibilidad de realizar cambios de forma automática.

Se asocian dos conceptos claves en la estrategia, los ***Semantic Wiki Bad Smells*** y los **Semantic Wiki Refactorings**. El primer concepto representa un error en la estructura semántica, mientras que el segundo representa el proceso para eliminar el error. De esta forma cuando se detecta un ***Semantic Wiki Bad Smell*** se tiene asociados cuales son los ***Semantic Wiki Refactorings*** que representan el proceso para eliminar el error. Logrando asistir al usuario en la mejora continua de la *ontología* con información necesaria para la toma de decisiones en todas las etapas de la evolución.

Se presenta el catálogo de ***Semantic Wiki Bad Smells*** que posibilita al usuario entender cuales son los errores que se deben analizar en la *ontología* de la ***Wiki Semántica***. Para cada ***Semantic Wiki Bad Smell*** se describe el error que representa, se describe un mecanismo de detección, se asocia un conjunto de ***Semantic Wiki Refactorings*** que se deben tener en cuenta para eliminar el error y se da un ejemplo para una mejor interpretación.

Se presenta el catálogo de ***Semantic Wiki Refactorings*** que posibilita al usuario eliminar errores de la *ontología* de una ***Wiki Semántica***. Se divide el catálogo en dos secciones: *Category Refactoring* y *Properties Refactoring*. Mediante la utilización de ***Semantic Wiki Refactorings*** se pueden eliminan todos los errores representados por los ***Semantic Wiki Bad Smell***. Se presenta cada ***refactoring*** con una descripción, los parámetros que son requeridos para la ejecución, se describe como es la implementación y finalmente se da un ejemplo para una mejor interpretación.

La herramienta **Semantic Evol** se implementa como extensión de ***Media Wiki*** para ser utilizada con la extensión ***Semantic Media Wiki***. El objetivo de la herramienta es asistir al usuario en el manejo de información semántica. Por las características del problema los usuarios de la herramienta son los mismos usuarios de las

***Wikis***, sin embargo se puede tener el perfil de un administrador de ***Wiki***.

Se tiene como objetivo que los mismos usuarios de las ***Wikis*** sean los que utilicen **Semantic Evol** por este motivo se propone una interface similar a la ***Wiki***. Se accede a la herramienta a través de las '*páginas especiales*' de la ***Wiki,*** y se propone una interface sencilla de utilizar. La definición de instancias de ejecución de ***refactorings*** también es definida de la misma manera que se crea un artículo.

Siguiendo con la filosofía de las ***Wikis*** se propone una implementación flexible y extensible. Es posible definir nuevos ***Semantic Wiki Refactorings*** y nuevos ***Semantic Wiki Bad Smells***. La herramienta **Semantic Evol** da la posibilidad de crear nuevos ***refactorings***. Debido a su implementación es posible definir nuevos ***refactorings*** para ser utilizados en una ***wiki semántica***. También es posible lograr la composición de ***refactorings***, logrando así un nuevo ***refactoring***. La herramienta **Semantic Evol** brinda la posibilidad de crear nuevos ***bad smells***. Debido a su implementación es posible definir nuevos ***bad smells*** para ser utilizados en una ***wiki semántica***.

Se pretende asistir al usuario durante todas las etapas de evolución de la *ontología*. La herramienta propone acceder a la información semántica a través de consultas semánticas. Durante las primeras etapas de la evolución de una ***Wiki*** es posible que no exista demasiada información semántica. En esa etapa puede suceder que no exista un criterio semántico que agrupe a los *artículos wikis* que el usuario requiere editar. El **Agrupador Semántico Web (*Semantic Grouper*)** permite realizar una o más consultas semánticas y agregar información semántica, de forma automática, a los artículos resultantes. Los usuarios ejecutan una serie de consultas semánticas, se obtienen como resultado *artículos wiki* que luego son seleccionados de forma manual por los usuarios. Finalmente la herramienta agrega información semántica solo a aquellos artículos que el usuario selecciona. Los usuarios tienen la capacidad de realizar una consulta de *artículos wiki* por extensión, determinando de forma manual cuales son los artículos que deben formar parte del conjunto. Luego estos artículos podrán ser seleccionados por una consulta semántica logrando así mayor flexibilidad para la herramienta.

La motivación de los usuarios para la generación de información semántica es el principal objetivo, brindando herramientas para mejorar la experiencia en la utilización de tecnologías semánticas es la mejor manera de lograr mayor participación de los usuarios. Se proveen conceptos e implementaciones sencillas que promueven la utilización y mejora continua de las ***Wikis Semánticas***. A continuación se describen cuales son los trabajos que se deberían realizar para continuar con la asistencia al usuario en la evolución en ***Wikis Semánticas***.

**2. Trabajo a futuro.**

Con el objetivo de asistir al usuario en la evolución en ***Wikis Semántica*** la principal necesidad es la investigación y desarrollo de nuevos ***Semantic Wiki Bad Smells*** y ***Semantic Wiki Refactorings***. Para lograr conceptualizar nuevos errores en la estructura de las ***Wikis Semánticas*** se deben definir nuevos ***Semantic Wiki Bad Smells***. Luego, a partir de los errores detectados se deben definir nuevos ***Semantic Wiki Refactorings*** con los procesos necesarios para eliminar los errores.

Los ***refactorings*** provocan cambios en la *ontología* de una ***Wiki Semántica*** de forma automática. Sin embargo, una vez que los usuarios realizan una modificación no es posible realizar las operaciones inversas de forma automática. Es decir, dejar sin efecto la ejecución de un ***refactoring*** regresando al estado anterior a la ejecución. Cuando un usuario tiene la necesidad de eliminar los cambios realizados por un ***refactoring*** debe hacerlo en forma manual. Este tipo de tareas muchas veces termina con la incorporación de nuevos errores en la *ontología*.

Se requiere la capacidad de deshacer o eliminar los cambios realizados por un ***refactoring***. Esta tarea se la conoce como ***UNDO*** (o deshacer). Consiste en intentar llevar a la *ontología* al estado anterior a la ejecución. Se debe discutir el concepto de estado de una *ontología,* cuales son los factores que influyen en el estado. Por ejemplo, que sucede si requerimos realizar un ***UNDO*** que afecta un *artículo wiki* que fue editado.

Con el objetivo de implementar el ***UNDO*** se debería analizar cuales son los casos en los que es posible realizarlo. Dada una *ontología* se debe analizar cuál es el estado antes de la ejecución, luego durante la ejecución realizar un historial de las acciones básicas realizadas por el ***refactoring***. Para cada una de las acciones básicas que el ***refactoring*** realiza, debemos definir cuál es la acción inversa a la misma. Por ejemplo, si una acción básica es agregar una anotación a un *artículo wiki*, la acción inversa es eliminar esa anotación. Si una acción básica es eliminar una relación de subcategoría, la acción inversa es agregar la relación de subcategoría. Finalmente se debería analizar cuál es el estado de la *ontología* al momento de la ejecución del ***UNDO***, dado que no siempre es posible llegar al estado anterior a la ejecución debido a modificaciones ejecutadas entre el momento de la ejecución y el momento en el que el usuario requiere el ***UNDO***.

La definición de un ***refactoring*** de acuerdo a la implementación, es a través de un *template* que es un *artículo wiki*. Mediante la exportación de artículos los usuarios de la extensión puedan compartir sus definiciones de ***refactorings*** y ***bad smells***. La comunidad de ***SMW*** puede compartir definiciones de ***refactorings*** y ***bad smells***.

Debido a la política de las ***Wikis*** no es conveniente eliminar un artículo de forma directa. Cuando **Semantic Evol** tiene la necesidad de eliminar un *artículo wiki* se realiza una eliminación lógica. La eliminación lógica consiste en eliminar el contenido del artículo, y agregar un comentario indicando que el artículo fue eliminado. Si la herramienta eliminara de forma directa los artículos que se solicita, puede suceder que otro usuario vuelva a crear nuevamente el artículo y así provocando el mismo error que llevó a la eliminación del mismo.

En este contexto resulta necesario una herramienta automática que elimine definitivamente los artículos que fueron eliminados lógicamente. Se propone el nombre de *Garbage Collector* por ser un comportamiento similar a los gestores de memoria de los lenguajes de programación. El administrador de la herramienta debe poder determinar un tiempo después del cuál los artículos son eliminados de forma definitiva. El tiempo será dependiendo del nivel de actividad de la ***Wiki.*** De esta forma se logra que la eliminación total de los *artículos wikis* sea de forma automática.

Se puede extender el comportamiento del *Semantic Grouper* para realizar la desagrupación de los conjuntos que se forman. De esta manera podríamos obtener una herramienta para administrar conjuntos de forma dinámica. En la actualidad la información semántica que genera el *Semantic Grouper* queda en los artículos hasta que los usuarios la eliminen. Puede suceder que esa información adicional sea de utilidad y represente un conjunto real, o que la información quede sin ningún objetivo.

Se propone el desarrollo de un historial de ***refactorings*** aplicados. El objetivo es entender el estado actual de la ***Wiki*** analizando cuales fueron los ***refactorings*** aplicados. Se debe contar con un listado ordenado por fecha de ejecución de todos los ***refactorings*** aplicados. Esta información también puede ser utilizada para realizar un análisis de los resultados obtenidos al aplicar ***refactorings***. De esta manera se pueden evaluar los beneficios de aplicar cada uno de los ***refactorings***. De forma conjunta, se puede indicar dentro del historial, cuál fue el ***bad smell*** que se detectó y provocó la aplicación de cada ***refactoring***. Esta información puede ser utilizada para el posterior análisis de las decisiones tomadas.

Se describen cuales son los desarrollos necesarios para continuar con la estrategia de asistencia al usuario en la evolución semántica. Las herramientas tienen el objetivo de asistir al usuario para brindar la mayor cantidad de información a la hora de tomar decisiones durante la evolución.

**Referencias.**


1. Leuf, B., Cunningham, W.: The Wiki way: quick collaboration on the Web. Addison-Wesley (2001).
2. Schaffert, S., Bry. F., Baumeister, J., Kielse, M.: Semantic Wikis. IEEE Sotfware 25(4) (2008) 8-11.
3. Fowler, M.: Refactoring: Improving the Design of Existing Code. Addison-Wesley, Boston, MA, USA (1999).
4. Semantic Wiki Refactoring. A Strategy to Assist Semantic Wiki Evolution. Martin Rosenfeld, Alejandro Fernández, and Alicia Díaz.
5. Semantic Media Wiki (SMW) Markus Krotzsch, Denny Vrandecic, and Max Volkel http://http://semantic-mediawiki.org/
6. Media Wiki. Http://mediawiki.org/
7. Hitzler, P., Krotzsch, M., Rudolph, S., Foundations of Semantic Web Technologies.
8. Semantic Forms. http://www.mediawiki.org/wiki/Extension:Semantic_Forms
9. IkeWiki http://semanticweb.org/wiki/IkeWiki
10. World Wide Web Consortium (W3C) http://http://www.w3.org/
11. Dbpedia http://dbpedia.org/

## Apéndice I.

### Instalación de la herramienta.

**Requerimientos:**

Para instalar la herramienta se requiere el siguientes software:

- Apache 2.2.17
- PHP 5.3.4
- MySQL 5.1.53
- PHPMyAdmin 3.2.0.1

  Se incluye una copia de *WampServer* [[ http://www.wampserver.com/ ]] que instala el software requerido.

  Se recomienda utilizar el navegador *Firefox* versión 18.0.1 [[ http://www.mozilla.org/es-AR/firefox/fx/ ]] para utilizar la herramienta.

### Guía de Instalación.

1. Instalar ***Media Wiki*** utilizando el siguiente tutorial:

   http://www.mediawiki.org/wiki/Installation

   Para realizar la instalación descomprimir el archivo '*mediawiki-1.19.3.tar*' en el directorio web (es recomendable cambiar el nombre a la carpeta una vez descomprimida, renombrar como '*mediaWiki*'). Acceder a través del navegador web hasta la ubicación y seguir las indicaciones para concluir la instalación. Es recomendable dejar todas las configuraciones predeterminadas a fin de probar la herramienta.

2. Instalar ***Semantic Media Wiki*** utilizando el siguiente tutorial:

   http://www.semantic-mediawiki.org/wiki/Help:Installation

   Para realizar la instalación descomprimir el archivo '*SemanticMediaWiki1.7.1*' en la carpeta 'mediaWiki/extensions' (es recomendable cambiar el nombre de la carpeta una vez descomprimida, en este caso se renombra como 'SemanticMediaWiki'). Luego agregar al archivo '*LocalSettings.php'* (que se encuentra junto al *index.php* de ***Media Wiki***) las siguientes lineas de código al final del archivo:

```
require_once( "$IP/extensions/Validator/Validator.php" );
include_once( "$IP/extensions/SemanticMediaWiki/SemanticMediaWiki.php" );
enableSemantics('example.org');
```

Luego, ingresar a la *wiki* y logearse como administrador (el usuario y contraseña se da en la instalación de la misma).

Ir a '*Paginas Especiales'* → 'Funciones de administración para Semantic Media Wiki' y ejecutar '*Iniciar o actualizar tablas*' y luego en '*Comenzar actualización de datos'*.

Finalmente, comprobar que la extensión está correctamente instalada agregando una anotación semántica a la wiki.

**3.** Instalar la extensión ***Semantic Evol***.

Para instalar la herramienta copiar la carpeta '*SemanticEvol*' dentro de la carpeta 'mediaWiki/extensions'.

Luego, agregar al archivo '*LocalSettings.php*' las siguientes lineas de código al final del archivo:

```
include_once("$IP/extensions/SemanticEvol/SemanticEvol.php");
$smwgNamespacesWithSemanticLinks[NS_TEMPLATE] = true;
$wgAllowExternalImages= true;
```

Luego, en el mismo archivo setear la variable $wgEnableUploads a 'true', la variable esta definida en 'false' en la linea 74. Se debe ver de la siguiente manera:

```
$wgEnableUploads  = true;
```

Finalmente, de forma opcional se pueden subir a la wiki los archivos de imágenes ubicados en la carpeta 'imagenes' del dvd adjunto.

**4.** Importar archivos de semántica.

Ir a '*P*aginas Especiales' → 'Herramientas de páginas' → 'Importar Páginas'. Luego, subir el archivo XML 'Semantic+Evol-version3-00Archivo1' y el archivo 'Semantic+Evol-version3-00Archivo2'.

Ir a 'Paginas Especiales' → 'Funciones de administración para Semantic Media Wiki' → 'Reparación de datos y actualización'. En este punto se requiere que los datos semánticos importados por la herramienta se actualicen para el correcto funcionamiento. Se recomienda navegar por los artículos de la categoría 'InlinequeryBadSmell' para forzar la actualización de los datos. Para comprobar si los datos fueron correctamente actualizados podemos ejecutar la siguiente consulta semántica desde 'Páginas especiales' → 'Busqueda semántica':

```
[[Category: BadSmellExecute]] [[Category:InlinequeryBadSmell]]
```

Esta consulta debe retornar tres artículos, en ese momento la información semántica está correctamente actualizada.

**5.** Subir las imágenes.

En la página principal de la Wiki, en el menú de Herramientas situado a la izquierda, ir a 'Subir un

archivo'. Ir a 'Examinar' y subir las imágenes de la carpeta 'Imagenes' del dvd adjunto. La herramienta ya queda instalada, se puede acceder desde 'Paginas Especiales' → 'Otras páginas especiales' → 'Semantic Evol'.

**Casos de Prueba.**

Se proveen casos de prueba para importar en la wiki. Se debe ir '*paginas especiales'* y luego 'Importar páginas' e importar los diferentes archivos de la carpeta 'Casos de Prueba' del dvd adjunto. Se debe tener en cuenta que se pueden sobreescribir algunas páginas, por lo cual es deseable comprobar que el estado inicial sea el deseado. Los ejemplos provistos son los mismos que se describen en esta tesis en cada una de las secciones de los ***rafactorings*** y ***bad smells***.

***Contacto.***

***Mauricio Hernán Etchevest.***

Email: mauricio.etchevest@gmail.com ; mauricioet@gmail.com

*cel: 0221 15 547 2225*